DOCUMENT RESUME

ED 296 710 IR 013 393

AUTHOR	Hunka, S.
TITLE	Computer Assisted Instruction Authoring System. Final Report. Appendix A: Photographs of Some Screen Displays. Experimental Authoring System (XAS). Appendix B: Author Manual for the DERS Interactive Experimental Authoring System (XAS). Appendix C: Syntax of Implementation Two. Computer Assisted Instruction Authoring System.
INSTITUTION	Alberta Univ., Edmonton. Div. of Educational Research Services.
SPONS AGENCY	Social Sciences and Humanities Research Council of Canada, Ottawa (Ontario).
REPORT NO	RIR-87-5; RIR-88-2; RIR-88-3
PUB DATE	6 Feb 88
NOTE	363p.; Photographs will not reproduce well.
PUB TYPE	Guides - Non-Classroom Use (055) -- Reports - Research/Technical (143)
EDRS PRICE	MF01/PC15 Plus Postage.
DESCRIPTORS	*Authoring Aids (Programing); *Computer Assisted Instruction; Computer Graphics; Computer System Design; *Courseware; *Minicomputers; Programing; *Programing Languages; Screens (Displays); *Specifications

ABSTRACT

This project report details the design of an interactive authoring system for the development of computer assisted instructional (CAI) software. This system is possible because the development of more powerful computing and software systems has facilitated authoring systems which allow the development of courseware in an interactive mode and without formal programming. The major CAI programming languages and emerging authoring systems are reviewed first, and specifications for the design of authoring systems are noted. Included in these specifications are those used in the development of this system, as well as those of a more general nature related to subsystems that support the authoring environment. The design and implementation of an interactive authoring system for the DEC VAX series of minicomputers is then detailed, including the creation of instructional graphics and instructional templates, and recommendations are made for the improvement of authoring system designs. The text is supplemented by various figures, and a 67-item bibliography is provided. Appendix A provides photographs of some of the screen displays of the experimental authoring system; Appendix B is a separate manual for the author using the system to create courseware; and Appendix C presents the syntax of the control, content, display, answer, input, and menu languages and expressions syntax used in the second implementation of the authoring system. This syntax is written in the Elf language, which was designed and developed by Alan Davis of the Division of Educational Research Services (DERS) at the University of Alberta. (EW)

ED296710

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)
☑ This document has been reproduced as received from the person or organization originating it.
☐ Minor changes have been made to improve reproduction quality.
● Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

research and information report

RIR-88-2

COMPUTER ASSISTED INSTRUCTION AUTHORING SYSTEM

Final Report

S. Hunka

division of educational research
faculty of education
the university of alberta
edmonton, canada

IR013393

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
S. Hunka
TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

RIR-88-2

COMPUTER ASSISTED INSTRUCTION AUTHORING SYSTEM

Final Report

S. Hunka

This Project was supported by a Grant from the
Social Sciences and Humanities Research Council
of Canada

Project No: 410-84-1111-R2
February 6th 1988

Division of Educational Research Services
Faculty of Education
University of Alberta
Edmonton, Alberta, Canada
T6G 2G5, (403)-432-3762

Acknowledgements

This project was supported by the Social Sciences and Humanities Research Council of Canada, project number 410-84-1111-R2. This support has been greatly appreciated.

A special thanks is extended to T. Garraway, and W. Chui for their design and programming efforts on the second implementation of the authoring system, and A. Davis for his contributions through Elf, and J. Hunka for assistance in testing various components of the system.

Summary

The development of quality courseware has been, and continues to be expensive because it is labour intensive. The shift away from using programming languages to authoring systems is one aspect which could facilitate cost reductions.

Although programming languages for the creation of courseware may provide in some instances the capability to create powerful features, they are seldom closely related to the instructional moves conceived of by an instructor. Thus specialized programming support is usually required. One major goal of an authoring system is to interface the author, in the simplest manner possible, to complex computing functions required to create the courseware. The domain of authoring action is however, extremely broad as they touch upon many computing complexities which are evident as single systems in their own right, e.g., text processors, graphics, plotting, understanding natural language input, etc. It is for this reason we doubt that a single authoring system without tailored modules will ever satisfy the needs of all authors.

A number of general characteristics of authoring systems are identified in the report, as well as features related to specific authoring actions. In the latter case they include authoring the screen using text, graphics, and plotters, student response modes, answer judging, arithmetic operations just to note the major ones.

The first implementation of an authoring system was made to explore the author-machine interface. This implementation was completed and used to create courseware which is in use. A manual (Appendix B) was produced and used. The major problems identified include the definition of the execution and edit space, and memory load placed on the author. Chapter 14 describes the first implementation and makes recommendations. The second implementation was designed and programmed, but not to the point where it contained sufficient features to use for the production of real courseware. However, the second implementation has a greater potential than the first implementation. The syntax of this implementation is given in Appendix C.

Table of Contents

Page

CHAPTER 1 1

INTRODUCTION

PROJECT OBJECTIVES 1
1.1 Review Of Authoring Languages 1
    1.1.1 Tutor 2
    1.1.2 Coursewriter II 3
    1.1.3 The CAN Authoring Language 3
    1.1.4 NATAL 4

    1.1.5 Digital Equipment

    1.1.6 Dimension Authoring Language (DAL) 5
1.2 Authoring Languages And The Authoring Process 6

CHAPTER 2 7
EMERGING SYSTEMS FOR AUTHORING 7
2.1 Authoring System Descriptors 7
    2.1.1 Response Analysis 8
    2.1.2 Branching 9
    2.1.3 Edit Capabilities 9
    2.1.4 Screen Use 10
    2.1.5 Author Prompting 10
2.2 Major Commercial Systems 10
    2.2.1 WICAT's Interactive System For Education (WISE) 11
    2.2.2 Tutor Based Authoring Systems 12
    2.2.3 TICCIT 12
    2.2.4 The ADAPT Authoring System 13
    2.2.5 The EASE Authoring System 13
    2.2.6 An Apple-based Authoring System 13
    2.2.7 CBT+ 13
    2.2.8 Course of Action 14
2.3 Characteristics Not Described 14
2.4 Summary 15

CHAPTER 3 16
SPECIFICATIONS OF AUTHORING SYSTEMS 16
3.1 The Authoring Process 16
3.2 General Characteristics Of The Authoring Environment 17
    3.2.1 Flexibility 17
    3.2.2 Minimize Author Memory Load 17
    3.2.3 Adaptation To Level Of Expertise 18
    3.2.4 Breadth Of Instructional Strategies 18
    3.2.5 Model Of Curriculum Content 18
    3.2.6 Interaction With Peripheral Processes 18
    3.2.7 Fit To Run-Time Environment 18
    3.2.8 Minimization Of Errors 18
    3.2.9 Fit To The Debug Environment 19
    3.2.10 Veridicality Of Authoring Actions 19
    3.2.11 Retreating From A Process And Error Recovery 19
    3.2.12 Help Systems 19
    3.2.13 Windows And Menus 19
    3.2.14 Special Terminal Functions 20
    3.2.15 Courseware Optimization 20
    3.2.16 Formal Testing Systems 20

    3.2.17 Access To External Utilities 20
    3.2.18 The Editing Process 20
    3.2.19 "What You See Is What You Get" 21
    3.2.20 Providing For Continuity 21
    3.2.21 Backup Services 21
3.3 The Generality Of An Authoring System 21
3.4 Software Maintenance 22
3.5 Human Factors in Interface Design 22

CHAPTER 4 25
THE STUDENT MODEL PROBLEM 25
4.1 A Broad Student Model 26
4.2 Examples of External Information 26
4.3 Examples of Internal Information 26
4.4 The Purpose of a Student Model In CBE 27
4.5 The Use of External Information 28
4.6 The Use of Internal Information 28
4.7 Analysis Of Information 30
4.8 The Problem of What Should be Modelled 30

CHAPTER 5 31
THE AUTHOR-SYSTEM INTERFACE 31
5.1 Introduction 31
5.2 Level Of Prescriptiveness 31
5.3 The Course Structure Problem 31
5.4 Windows 33
5.5 Menus 33
5.6 Icons 34
5.7 Flow Charts 34
5.8 Informing And Helping The Author 35
5.9 Warning and Error Messages 36
5.10 Author Entry Mode 36
    5.10.1 Keyboard Entry 36
    5.10.2 Pointing Responses 37
    5.10.3 Bit Pad 37

CHAPTER 6 38
AUTHORING THE SCREEN
TEXT 38
6.1 Introduction 38
6.2 Text Parameters 38
6.2.1 Screen Characteristics 38
6.3 Font Type And Style 39
6.4 Character Positioning 39
6.5 The Use Of Color 40
6.6 Character Size 40
6.7 Dynamic Text Formatting 40
6.8 Text Rotation 41
6.9 Display Of Stored Information 41
    6.9.1 Screen Arithmetic 42
6.10 Other Features 42

CHAPTER 7 43
AUTHORING THE SCREEN

GRAPHICS 43
7.1 Introduction 43
7.2 The Hardware Interface Problem 44
7.3 Graphics In Support Of Text 44
7.4 Simple Graphics 44
7.5 Graphics And Special Fonts 44
7.6 Plotting Data 45
    7.6.1 Defining The Axes 45
    7.6.2 Plotting Characteristics 46
    7.7 3-Dimensional Plotting And Rotation 46
7.8 Animation 46

CHAPTER 8 47
TEXT AND GRAPHICS EDITORS 47
8.1 Introduction 47
8.2 Compromises And Issues 47
8.3 The Edit Space 48
8.4 Text Editing 48
    8.4.1 Text Positioning 49
    8.4.2 Color Changes 49
    8.4.3 Size Changes 50
    8.4.4 Slope Changes and Rotation 50
    8.4.5 Editing Special Control Characters 50
    8.4.6 Editing Overstruck Characters 50
8.5 Graphics Editing 50
8.6 Editing Plotted Graphics 51
8.7 Multiple Editors Vs A Single Editor 51

CHAPTER 9 52
AUTHORING THE STUDENT RESPONSE MODE 52
9.1 Introduction 52
9.2 The Nature Of The Response 52
9.3 Identification Of Student Input 53
9.3.1 Definition Of Keyboard Input 53
9.4 Defining A Pointing Input 54
9.5 Graphical Input 55

CHAPTER 10 56
AUTHORING FOR ANSWER JUDGING 56
10.1 Introduction 56
10.2 The Nature Of The Input 56
10.3 String Input 56
10.4 Numerical Input 56
10.5 Pointing Input 57
10.6 Graphical Input 57
10.7 Typical Solutions 57
10.8 Processing The String Input 58
    10.8.1 Cleaning Up The Input String 58
    10.8.2 Transformations Of The Input 59
10.9 Defining The Standard For Comparison Of String Input 59
    10.9.1 Processing The Standards 60
10.10 The Transformation Report 60
10.11 The Matching Processes 62
10.12 Diagnose 63

10.13 Definition Of Evaluate Process ... 64
10.14 Display ... 65
10.15 An Example ... 65
10.16 Problems Related To REPORT And DIAGNOSE ... 67
    10.16.1 Report Issues ... 67
    10.16.2 Diagnose Issues ... 68
        10.16.2.1 Example 1 ... 68
        10.16.2.2 Example 2 ... 69
        10.16.2.3 Example 3 ... 69
        10.16.2.4 Example 4 ... 69
10.17 Analyzing Algebraic Expressions ... 70
    10.17.1 Spatial Analysis ... 71
    10.17.2 Expand To Word Boundary ... 71
    10.17.3 Evaluation Of Expression ... 72
10.18 Reviewing The Problem Of Analyzing Typed Student Responses ... 72
10.19 Evaluating The Pointing Response ... 73
10.20 Specifying The Standard ... 73
10.21 Response Analysis Related Variables ... 73

CHAPTER 11 ... 74
ARITHMETIC OPERATIONS ... 74
11.1 Introduction ... 74
11.2 Courseware Arithmetic Needs ... 74
11.3 Decision Making Needs ... 74
11.4 Authoring Needs ... 74
11.5 Limitation Of Arithmetic Capabilities ... 75
11.6 Arithmetic Variable Types And Forms ... 75
11.7 Arithmetic Operations ... 75
11.8 String Operations ... 75
11.9 Special Short Form Operations ... 76
11.10 Naming Convention ... 76

Chapter 12 ... 77
Interfacing the Author to the Authoring System ... 77
12.0 Introduction ... 77
12.1 Author Entry to the CAI System ... 77
12.2 Author Entry to the Authoring System ... 78
12.3 General Author Interfacing Considerations ... 79
12.4 Interfacing the Author to the Course Structure ... 81
12.5 Interface to Curricular Units ... 81
    12.5.1 Micro View of Course Structure ... 81
    12.5.2 Macro View of the Course Structure ... 82
12.6 An Integrated System of Information about the Course ... 84

Chapter 13 ... 85
Defining and Interrelating Functional Units ... 85
13.0 General Considerations ... 85
13.1 Veridicality ... 85
13.2 Functional Units of the Authoring System ... 85
    13.2.1 Entry to the Authoring System ... 85
    13.2.2 Starting the Authoring Process ... 86
    13.2.3 General Authoring Functions ... 87
    13.2.4 Text to Screen Functions ... 89
    13.2.5 Text-Graphic Functions ... 90

13.2.6 Graphic Functions.....90
13.2.7 Plotting Functions.....91
13.2.8 Displaying the Graphics.....94
13.2.9 Screen Erasure.....94
13.2.10 Store Functions.....95
13.2.11 Input Functional Units.....95
13.2.12 Input Transformations.....97
13.2.13 Answer Analysis Functional Units.....99
13.2.13.1 Logic Flow Chart.....106
13.2.14 Arithmetic Operations.....106
13.2.15 Templating.....108
13.2.15 Comment.....110
13.2.16 Run and Edit.....110
13.2.16.1 Authoring and Editing.....110
13.2.16.2 Executing and Editing.....113
13.3 Need for Research.....113

Chapter 14.....114
Implementation.....114
14.0 Introduction.....114
14.1 Entry to Authoring Confirming Information.....115
14.2 The Author's Main Command Menu.....119
14.2.1 Functions of The Main Commands.....120
14.3 A Typical Authoring Sequence.....126
14.4 The Nature of the Author Interface.....127
14.5 Specific Functional Units.....128
14.5.1 Screen Format.....128
14.5.2 Scale Option.....129
14.5.3 The Pattern Option.....129
14.5.4 Label Option.....131
14.5.5 The pr and prr Options.....132
14.5.6 The Text Option.....133
14.5.6.1 Suggestions and Recommendations.....136
14.5.7 Lines Option.....137
14.5.8 The Draw Option.....138
14.5.8.1 Suggestions and Recommendations.....140
14.5.9 The Plot Option.....140
14.5.9.1 Suggestions and Recommendations.....144
14.5.10 The Picture Option.....144
14.5.10.1 Suggestions and Recommendations.....144
14.5.11 The Erase Option.....145
14.5.11.1 Suggestions and Recommendations.....145
14.5.12 The Store Option.....145
14.5.12.1 Suggestions and Recommendations.....147
14.5.13 The Input Option.....147
14.5.13.1 Suggestions and Recommendations.....148
14.5.14 The Transform Option.....148
14.5.14.1 Suggestions and Recommendations.....148
14.5.15 The Answer Option.....149
14.5.15.1 Suggestions and Recommendations.....152
14.5.16 The Arithmetic Option.....153
14.5.16.1 Suggestions and Recommendations.....154
14.5.17 Comment Option.....155
15.5.17.1 Suggestions and Recommendations.....155

15.5.18 Branching Option....155
14.5.18.1 Suggestions and Recommendations....156
14.5.19 The Run Option....156
14.5.19.1 Suggestions and Recommendations....156
14.5.20 The Edit Option....157
14.15.20.1 Suggestions and Recommendations....157
14.15.21 The Exit Option....158
14.15.21.1 Suggestions and Recommendations....158
14.2 Generality of Authoring Systems....159

Chapter 15....161
Second Implementation....161
15.0 Introduction....161
15.1 General Hierarchical Structure....161
15.2 The Control Language....163
15.3 The Content Module and Language....163
15.4 Display Language....164
15.5 The Input Language....165
15.6 The Answer Language....166
15.7 Menu Language....167

Chapter 16....168
Future Research Considerations....168
16.0 Introduction....168
16.1 Areas in Need of Study and Research....168

Bibliography....171

Appendix A Color Photographs of Some Screen Displays
Appendix B First Implementation Author's Manual
Appendix C Syntax of Second Implementation

Figures

Page
Figure 4.1 Application of External and Internal Information in the Student Model 29
Figure 5.1 Linear and Tree Structure Representations 32
Figure 12.1 Different Representation of Course Structure 82
Figure 12.2 Diagrammatic Representation of Course Structure.. 83
Figure 12.3 Expanded Representation of Course Structure with References to Associated Information 84
Figure 13.1 Screen Related Functions 89
Figure 13.2 Line Drawing Functions 90
Figure 13.3 Draw Functional Units 91
Figure 13.4 Plotting Functional Units 93
Figure 13.5 Graphic Display Functional Units 94
Figure 13.6 Store Functional Units 95
Figure 13.7 Input Functional Units 97
Figure 13.8 Transformation Functional Units 99
Figure 13.9a Functional Answer Units 102
Figure 13.9b Sub Answer Unit FUA 103
Figure 13.9c Sub Functional Units FUB 105
Figure 13.10 Functional Arithmetic Units 108
Figure 13.11 Templating Functional Units 109
Figure 13.12 Example of Iconic Representation for Unobservable Component in Courseware 112
Figure 14.1 Authoring System as Part of CAI System 115
Figure 14.2 Initial Information Presented to Author 117
Figure 14.3 Main Authoring System Menu 119
Figure 14.4 Functional Units of the Pattern Option 130
Figure 14.5 Functional Units of the Label Option 132
Figure 14.6 Functional Units of the Text Option 134
Figure 14.7 Functional Units of the Lines Option 137
Figure 14.8 Functional Components of the Draw Option 139
Figure 14.9 Functional Units of the Plot Option 142
Figure 14.10 Functional Units of the Store Option 146
Figure 14.11 Functional Units of the Answer Option 150
Figure 14.12 Functional Units of the Arithmetic Option 153
Figure 14.13 Modular Component for Selection to Meet Authoring Needs 160
Figure 15.1 Hierarchical Course Structure 161
Figure 15.2 Functional Scope Space at Level n 162
Figure 15.3 Calling Domain of the Content Module 164

CHAPTER 1

INTRODUCTION: PROJECT OBJECTIVES

The original project objectives were as follows:

1. An examination of existing authoring languages such as Tutor, Natal, Can, and Coursewriter was to be made to determine the types of actions which would be required in an authoring system.

2. An examination of a variety of instructional strategies operational in CAI courses at the University of Alberta to determine what authoring actions would be required in an authoring system.

3. Design and implement an authoring system which would allow an author to construct instructional computer programs for a wide variety of subject matter, student characteristics, and instructional strategies.

4. To test the efficacy of the authoring system developed.

1.1 Review Of Authoring Languages

Programming languages, including authoring languages, represent to a great extent a set of precisely defined operations, which when properly ordered produce the desired processes when executed. The variety and power of a set of basic operations may determine the richness of the processes which may be embodied in the final product. However, "variety" and "power" of the operations must be judged relative to the needs of the author during the definition of instructional sequences, and their execution. An authoring language, although it may have many defined operations from which a large variety of instructional processes may be constructed, may also require very detailed specification of elementary operations, and specialized skills for their use, resulting in high programming and maintenance costs. On the other hand, languages which are characterized by a small set of basic operations may provide for building powerful specific functions, but be very limiting or constraining in their support of a wide variety of instructional processes.

The multiplicity and variety of programming languages, as well as their continual enhancement, attests to the desire of computer users to have languages which span a broad range of specificity and applicability.

Programming languages cannot be solely characterized by the specificity of their basic operations. Many other factors may detract or enhance their usefulness. For example, a language may have very basic operations, but be most useful for the development of other, higher level languages; some languages allow extensive numerical operations, while others may have very limited numerical operations but extensive capabilities for manipulations of character strings. In addition, the environment in which a language is made operational and the support services which are available can also be very important to the usefulness of the language.

Any comparison of author programming languages has to recognize the range of capabilities of the language. Some languages make no attempt to provide for the creation of graphics, and for this reason may appear to have less complexity. Although, knowing whether a language can handle integer as well as real numbers, scalars, as well as arrays, may indicate to the computing specialist the capabilities of the language, for many instructors this may appear irrelevant. Their concern is with the ease with which the final courseware can be created.

In the final analysis, any programming language must be judged by the degree to which it provides the vehicle for allowing the user to efficiently produce an end product. In the case of instruction, the end product may require, depending upon the nature of the subject matter and the demands of the author, a very wide range of computing capabilities including graphics and animation, natural language understanding, numerical computations, information retrieval, and the generation of sound. It is unlikely that a single authoring language can efficiently and effectively accommodate such a spectrum of functions and still be efficiently used by an instructor.

1.1.1 Tutor

Tutor is the authoring language used on the PLATO system, originally developed at the University of Illinois. This authoring language (Sherwood, 1977, Avner and Tenczar, 1969, Control Data Corporation, 1978) contains 289 different commands. This number must be judged in the light of the capabilities which are provided, which include the creation of graphical displays.

Sixty-nine instructions are provided for purposes of calculations, arithmetically related operations, and handling of information. These include the following: *define, calc, add1, sub1, zero, calcc, calcs, pack, iota,find search, move, compute, clock, date, date, name, course, from, lessin, randu, setperm, randp, remove, modperm, common, comload, storage, stoload, block, and transfr.* Some instructions provide for arithmetic processes on values other than base 10, and sorting processes.

Seventy-seven instructions are available to control the display of information to the student. These instructions also include operations for the creation of graphical displays. The instructions are as follows: *at, write, writec, show, showo, shows, showe, showa, erase, size, rotate, dot, draw, circle, circleb, window, mode, catchup, delay, char, plot, charset, altfont, micro, codeout, tabset, slide, enable, disable, ext, origin, axes, bounds, frame, scalex, scaley, lscalex, lscaley, labelx, labey, markx, marky, locate, graph, hbar, vbar, vector, delta, funct, gdraw, and polar.* A number of these instructions are related to the display of numeric information.

Seventy-three instructions are available for control of lesson execution. Although these are considered to be part of the Tutor authoring language, their inclusion likely reflects more upon the non-modularity of the Tutor software system which is characteristic of earlier styles of software design and firmware capabilities. This set includes the following instructions: *unit, next, jump, join, do, goto, exit, nextnow, iferror, entry, finish, end, help, help1, data, data1, lab, lab1, back, kstop, base, term,* *,$$, *stop, start,* stop*, *restart, return, jumpout, inhibit, press, pause, force, change, and use.* A number of these instructions enable and disable certain keyboard functions available to the student.

Fifty-eight response handling instructions are available in Tutor. These instructions form the basis for interpretation and judging a student's input. These instructions include the following: *arrow, endarrow, long, jkey, storea, open, loada, close, bump, put, putd, answer, wrong, list, concept, vocab, ok, no, ignore, exact, exactc, ansva, ansv, wrongv, store, storen, match, touch, or, ans, specs, judge, edit, copy, and time.*

These instructions include string manipulations such as those required for "cleaning up" a student's input, as well as assessing the adequacy of numerical response.

Tutor has 10 instructions for the collection and storage of data generated during course execution. These instructions form the basis for performance recordings. The instructions include *dataon, dataoff, area, output, output1, readset, reada, and readl.* Three additional instructions, *dataset, datain, and dataout,* are used for the manipulation of data set files.

The large set of Tutor instructions allows for the creation of a large variety of processes including the following: the definition of integer and real values as scalars or arrays, common variables, the use of expressions and functions, logical and bit operations, calculations, string searching, calculated branches, graphing, and timing control.

There is little doubt that Tutor can be considered the most complex of all courseware programming languages.

1.1.2 Coursewriter II

IBM has had a number of versions of the Coursewriter authoring language, the most comprehensive being that version defined for the IBM 1500 system (IBM, 1970). (Part of the comprehensiveness of Coursewriter II was due to the development by the IBM 1500 users of specialized functions (Romaniuk & Schienbein, 1973) which are not formally defined in IBM's Coursewriter manual). The 1500 system contained a 16mm film projection and audio tape facility, which also required the definition of commands for control of these peripherals.

Coursewriter is characterized by the definition of "op codes" (operation codes) representing an abbreviated form of a command word. An example of an op code would be "dt" followed by a set of parameters. "dt" is to represent the words "display text". In this sense the authoring language is more characteristic of early and more general assembler-type programming languages.

Approximately 25 op codes were defined in Coursewriter. This number is approximately doubled with the addition of special modifiers. These codes with their meaning are as follows: *pr and prr (problem start and restart), ty (type text or contents of buffer), dt (display text), de (display erase), dl (display emphasis line), pm (proctor message), au (audio), fp (film projector), pa (pause), ep (enter and process), ec (enter and continue), ca (correct answer), cb (synonymous correct answer), wa (wrong answer), wb (synonymous wrong answer), aa (additional answer), ab (synonymous additional answer), un (unrecognizable answer), ea (end of answers), ad (add), sb (subtract), mp (multiply), dv (divide), ld (load), br (branch, conditional and unconditional), lr (load return register), cm (call macro), fn (function call), and, ma (macro call).*

1.1.3 The CAN Authoring Language

The CAN authoring language was originally developed at the Ontario Institute for Studies in Education (Olivier, 1975), and later enhanced somewhat at McGill University (Cartwright, 1978).

This authoring language has approximately 33 operation codes, and each code could have a modifier, and a varying number of terms or parameters. In this sense it has characteristics of Coursewriter, but unlike Coursewriter a single instruction frequently included are large set of parameters. The instruction set when categorized into types include those instructions used for display, response input and processing, numerical processing, program control, and file manipulation. In addition 100 variables (with fixed names) could be defined for use in arithmetic expressions. The major operation codes are as follows: T (text which include special control characters), PM (play message), SS (show slide) A (answer, get an input, check an answer, and branch to), U (unanticipated answer), C (correct), I (incorrect), W (wrong answer), M (mode conditions for answer matching), E (replace one string with another), RT (response time), RL (response length), RK (read keyboard), RI (reset input), CN (convert numeric), CO (compute), RV (random value), RC (read clock), G (go), S (subroutine call), R (return), CS (clear stack), DM (delete from stack), PA (pause), IF (conditional), RA (restart address), IA (interrupt address), END, X (execute), RU (record unanticipated), RD (record data), D (data), RF (read file). Two examples of code which an author (or programmer) would be expected to create would be as follows:

```
T,Type an expression or "End" to quit, c\r 909 A,@ E@ ; 9237 c/r
9237 U CO,V99,@A c/r T,=@V99 c/r G,909 c/r
```

The modifications reported by Cartwright included a Begin-End block to facilitate more structured programming, text modifiers (19 of them) to handle CRT displays, answer list modifiers, and 4 new opcodes: L (load buffer), KA (desk calculator mode), PS and PE for selecting test item questions. A new code, DEF, was used to redefine opcodes into more readable terms such as for example, the use of TEXT instead of the character T.

1.1.4 NATAL

NATAL (Brahan & Godfrey, 1984) is a distinctly Canadian development of an authoring language. The development of NATAL was sponsored by the National Research Council through its Associate Committee on Instructional Technology. The initial formal specification of NATAL was done by Westrom at IBM (Westrom, 1974). Initial implementation of the language was made on a DEC PDP 10 system at the NRC laboratories in Ottawa. Later, Honeywell was contracted to produce a version of NATAL for distribution. More recently a version of Natal for micro computers has been developed (Farley, et al, 1986). Dr. Godfrey at the University of Victoria, has been instrumental in the commercialization of NATAL and spearheading a NATAL users group.

NATAL represents as much a departure from Tutor as does Pascal from Fortran. NATAL is characterized by being a much more highly structured authoring language than Tutor or Coursewriter. The initial design of NATAL still envisaged the need to "code" a program in the same sense as Tutor or Coursewriter. The application of structured programming techniques, introduces a modularity of course design which is not present in other languages, e.g., the formal definition of procedures and units.

Programming in NATAL requires the use of an extensive set of keywords or commands, although perhaps not as extensive as in Tutor or Coursewriter. Examples of the types of commands which are used in NATAL are as follows: *all, append, at, attn, bin, by, call, caller, checkpoint, close, condition, continue, delete, disable, display, do, edit, else, enable, end, endcourse, endfile, entry, error, exit file, first, font, function, go, goto, graphic, hex, hold, if, input, insertion, keyboard, last, map, next, nchar, noreplace, on, on condition, otime, plot, posn, procedure, proctor, read, record, reference, reinf, release, rentry, repeat, response, restart, retry, return, right, rtype, scol, sedit, set, sfont, shareable, signal, stime, stop, syn, then, time to, true, type, unit, unrec, while, work, write, and wrong*. In addition to these statements, approximately 33 special symbol combinations are defined for the Display Sub-Language in order to control and format the screen display a student would see. NATAL in this regard has the elements of a text formatter as part of its language definition, similar to the text processor software currently available on many main frame computers.

1.1.5 Digital Equipment: Dimension Authoring Language (DAL)

Digital Equipment's Dimension Authoring Language (DAL) (Digital Equipment Corporation, 1982) looks very much like Control Data's Tutor language. A major difference between Tutor and DAL is that the latter requires a structured style of programming by the author. There are over 100 instructions in DAL, including a large number related to the handling of graphics. DAL also allows the definition of variables including arrays. The kinds of instructions which must be used in order to create courseware are described below.

Variables are handled through the instructions *assign, define, funct,* and *seed* , while screen writing and the control of graphics are handled by such instructions as *at, erase, write, rat, bcolor, fcolor, axes, box, circle, clip, curve, delta, dot, rdot, gat, gbox, gcircle, rcircle, gcurve, rcurve, gdot, gline, gorigin, graph, gvector, hbar, href, line, rline, lscalex, lscaley, scalex, scaley, markx, marky, mload, mode, mplot, pattern, rbox, rorigin, rsize, rvector, vbar, vector.*

The lessons can be divided into instructional units by the use of *unit, lesson, endlesson, test,* and *endtest,* and the control of logic by *branch, do, for, endfor, else, if endif, loop, endloop, pause, outloop, redo, reloop, return, stop, value,* and *when* .

Response processing is handled by such instructions as *convert, delimit, endq, judge, query, wrong, right weight, markup, match, score, noise, noword, okword, score, specs, rightv, syn,* and *wrongv,* while input and output by *close, get, include, log, open, eof, find, rewind, prompt, put, slide, tray.*

DAL also provides access to the control of fonts sets. The instructions used for this purpose are *char, charset, italics,* and *trotate*.

To make use of DAL, like Coursewriter and Tutor, an author would most likely require a programmer to translate instructional processes into program code.

DEC's Courseware Authoring System (CAS) (Reed & Porter, 1984) appears to be a collection of existing software tools only some of which were specifically designed for computer based instruction. Tutorial and drill templates are provided for creation of courseware. A graphics (for DEC's Regis graphics code) and font editor are available as well as a "slide" system for the presentation of graphics. Answer checking is similar to those procedures in Tutor, but unlike Tutor and its PLATO environment, interactive video can be used.

Digital Equipment also provides a run-time environment for its system. This run-time environment includes the definition of a hierarchical supervisory structure which includes a manager, the instructor, author, and students. Some progress reports are available, as well as a browse mode and a mail facility.

1.2 Authoring Languages And The Authoring Process

Although a cursory comparison can be made among the more prominent authoring languages, this comparison can only suggest the complexity of translating instructional paradigms into instructional courseware. If an author were to use any one of the authoring languages discussed above, clearly a large amount of precise information would have to be meticulously applied in order to produce courseware, or the task of creating the courseware would have to be supported by a team which would include professional programmers.

However, noting all the programming codes into which course content and instructional strategies must be integrated does not present a complete representation of the authoring problem. Different authoring languages may have widely differing capabilities, and therefore considerable difference in the extensiveness of their programming codes. For example, if an authoring language includes the authorship of graphics, control of special peripheral devices, extensive monitoring of student progress, or a host of support services for the student and the instructor, then the number of programming codes required is likely to be higher than for a language which allows the creation of simple drills or test items. A specific example in this context is that Coursewriter II does not contain the capability for the creation of graphics, while Tutor does; Coursewriter II provided control over an audio tape peripheral, while Tutor does not. (Control of a microfiche projector and a Sony cassette device, and later a video disk were provided for the PLATO system at the University of Alberta, but these were local developments and not supplied with the PLATO system by Control Data Corporation).

An examination of the authoring languages makes it obviously clear that the syntax is only remotely related to the activity of instruction, perhaps even less so than, say Fortran is to numerical computation! It is likely that this unfortunate phenomenon reflects the lack of early input by instructors to the process of author language development and its evolution.

The problem is compounded exponentially when one considers that the instructional activity is frequently poorly bounded, and frequently calls for the utilization of computing procedures which in themselves are still research specialities in the discipline of computing science. For example, instructors would like an instructional system to have an understanding of natural language input, graphics comparable to those created for the motion picture industry and CAD/CAM systems, symbolic manipulation of algebraic functions, easy organization and access to data bases, and so on. Each of these is today the subject of much research in computing science. On this basis alone, one must expect that the development of authoring procedures will continue to evolve, and that no one system is likely to become a standard with any degree of maturity and stability in the foreseeable future.

CHAPTER 2

EMERGING SYSTEMS FOR AUTHORING

Wolman (Wolman, 1986) has provided a brief documentation of over one hundred "authoring systems". The efficacy of these authoring systems is likely reflected in the fact that their listed prices range from a low of a few hundred dollars to tens of thousands of dollars. Any brief description of their capabilities is likely to be incomplete. The only adequate test of such software is its application, clearly a formidable task given the number of such systems, the costs of acquiring the systems and the variety of hardware on which they operate, and the human time required to explore the capabilities of each.

2.1 Authoring System Descriptors

Major categories used in the documentation provided by Wolman include the following:

1. Response Analysis
2. Branching
3. Editing
4. Prompting
5. Screen Use
6. Graphics, and
7. Hardware required as well as thirty-six other categories related to the more general context of authoring. These other categories include the following:
    a. tracking of student progress
    b. on-line course listings and student registration
    c. interface with programming languages and other software
    d. interface with videodisc and videotape
    e. use of mouse, light pen, and touch sensitive screen
    f. multiple input on single screen
    g. superimpose text over graphics
    h. batch input of files
    i. editor for unresolved branches
    j. random test generator
    k. color, and sound
    l. lower case characters, variable character sets
    m. animation
    n. multiple correct answers, close answers
    o. separate presentation system
    p. networking, uploading to mini or mainframe
    q. authoring on micro, presentation on micro
    r. vendor support services

Although these categories reflect facilities upon which some distinctions can be made among the systems, one must make only inferences about the nature of the instructional paradigms which may be created. Noting the characteristics which are used to describe various "authoring systems" should suggest to the reader the wide heterogeneity in what is considered to be useful capabilities.

2.1.1 Response Analysis

An examination of the descriptions under this category suggests that many authoring systems are directed towards rather specific instructional applications, such as for example, industrial training and testing, and that the author is constrained, perhaps for sake of keeping a system easy to use, to selected instructional strategies.

A review of the descriptors in the category of response analysis, will suggest the variety of directions taken, the needs of some authors and the constraints of some systems:

1. responses per screen; characters per response; characters per question; single or multiple character input. These descriptors suggest a tightly bounded input format for the learner under some systems.
2. number of multiple choices and items in matching type questions; fill in blanks; free response. Some systems appear to be designed for simplicity of asking questions and analyzing response, while others provide considerable freedom of student input, but with the associated problems of the software "understanding" that input.
3. edit of student's input prior to judging the adequacy of the input. Systems with highly restrictive input, e.g., a single character, minimize the problem of interpreting the student's input.
4. "wild" character; keyword; numerical range or value; punctuation and case sensitivity; use of logical operators; fuzzy matching; spelling check; list of words to be removed from input; use of true subscripts and superscripts; input of numerical expressions; pattern matching; alternative answer sets. A very wide range of response analysis procedures is suggested by this list. Some procedures may require, if any degree of generality is to be obtained, a fair amount of computational power, eg., fuzzy matching and pattern matching.
5. use of counters and switches; number of branches per screen and question; termination after n tries; timeout; These suggest that in some systems specific constraints exist in the logic following the presentation of a question.
6. defined reprise area in terms of lines or position on the screen; markup facility; prestored feedback. Some systems constrain the nature of the feedback and the area into which the feedback can be presented to the student.
7. support services such as special question debuggers or help features.

It should be noted that Control Data Corporation has developed entirely separate systems for different types of instructional paradigms, e.g., Advanced Tutorial Model, Certification Testing Model, Drill and Practice Model, Graphics Simulation Model, Situation Simulation Model, and Tutorial Lesson Model.

2.1.2 Branching

Authors must have the capability of defining branches explicitly, or understand the conditons under which implicit branching is made. The crux of the problem is the ease with which an author can make use of branching without the creation of errors. Descriptions of current authoring systems give little indication of how the author creates branches. The following descriptions suggest that most systems have a rather simple branching capability.

1. number of branches per frame
2. conditional branches
3. branching on student input and history
4. use of goto statements
5. user exit to external programs
6. forward and backward branching
7. based on logical, variable, or text
8. branches on wrong answer
9. use of student menus
10. branching associated with key presses
11. computed branch
12. table of branch points

This list suggests that some authoring systems are very much frame oriented. The definition of specific keys to effect branches by the student, suggests that the author must be able to relate "PF-type" keys to specific branches. The need to allow an author to easily and accurately define branching conditions presents a difficult problem in the design of authoring systems and man-computer interfaces.

2.1.3 Edit Capabilities

The definition of a system for editing the author's work becomes a complex problem when a variety of instructional techniques are employed. In simple authoring systems editing systems allow the correction of textual information and the structures which relate one frame display to another. However, when the author has the capability of using graphics, audio, simulation techniques, and complex branching definitions, the editing of the courseware is no minor problem. This problem is difficult to handle because different data structures are usually used to store textual, graphical, and audio information, each requiring different editing algorithms.

The descriptors used to describe the editing capabilities of authoring systems noted by Wolman are as follows:

1. color changes, insert and delete
2. line editor; line insert and delete
3. use of external editor
4. graphics editor; video and sound editing
5. highlight statements in error for editing
6. window size changes and position
7. edit character fonts
8. cut and paste
9. word processing-like editor
10. menu prompted editor

11. immediate author try out
12. text and question editor
13. build displays as student would see them
14. interactive graphics editor

2.1.4 Screen Use

Descriptors in this category suggest the capabilities available in the use of the display screen.

1. lines per question; answers/question
2. window manager
3. areas for defined activities
4. graphics and text can be mixed
5. temporary work areas

Some authoring systems appear to have rigid specifications defining the function for which areas of the screen can be used. In some systems, text and graphics cannot be mixed.

2.1.5 Author Prompting

Many of the authoring systems described by Wolman appear to make use of menus.

1. templates
2. "one finger authoring", presumably special function keys
3. menus contain only currently available options; context dependent
4. prompt may be turned on and off
5. pop up/down menus
6. command lines
7. letter or numbered options

The adequacy of menus depends very much upon the context of the user interface and the nature of the implementation. Their use does provide some degree of control of authoring errors by presenting only those options which are tenable at a particular point in the authoring process.

2.2 Major Commercial Systems

A number of systems for the creation of courseware have been developed by the larger computing companies. Some of these have been referred to as authoring systems, perhaps more for purposes of marketing than for providing an accurate descriptions of the facilities available.

2.2.1 WICAT's Interactive System For Education (WISE)

Descriptions of this system suggest that it is one of the few systems which has been specifically designed for authoring purposes. It appears to be more than a collection of tools initially designed for purpose other than authoring.

The WISE system has two major components: an authoring component, and a run-time component. (A third component is a run-time component for delivery by a specific micro computer.) WISE appears to be a system for producing "frame" oriented CAI, where each frame may contain course display features and/or course logic features. However, the term frame as used in the WISE system should not be taken to mean a highly restrictive unit of instruction. The authoring functions include lesson-logic creation, lesson display creation, response analysis, computation, testing and record keeping. Special editors (frame editors) related to these functions form the basis for author definition of courseware. The lesson-logic functions are developed using a variety of frame editors. Frame editors exist for the following functions: menu design, presentation, free response questions, multiple choice questions, calculations, external programs and lessons, algebraic expressions, function plotter, test management, and data collection. Other editors are defined for graphics and include editing for animation, video control, and windows.

Fairweather and O'Neal (Fairweather & O'Neal, 1984) have described the WISE authoring system in the context of characterizing authoring systems as to their power, ease of use, and productivity. The WISE system appears to be one of the best CAI authoring systems available. This system takes into consideration different levels of users by providing them with different levels of capability. An author may create courseware through a set of prompted menus, through a "calculation frame", essentially an "authoring language", and within the calculation frame, may access any program or utility supported by the computer's operating system. The latter provides capability to access other programming languages within the environment of the WISE execution system. The calculation frame provides access to computation, creation of displays, conditional branching, use of system variables, as well as to the creation of student input, answer judging, and control of external devices. A display editor allows the definition of special symbols or character sets down to the pixel level, as well as their manipulation in terms of size, and orienting transformations. Graphic objects are scaleable, rotatable, and moveable as well as being "animated" (deposited on a defined path). Student input may be defined within an area of the display screen, in any sized font and color. Any set of keys can also be defined as special function keys, of particular advantage for the definition of processes requiring many instructional steps. Answer processing includes keyword searches in the context of a list of words to be ignored and defined by the author in a special library of such words. Spelling checks and substring searches are also available. The author may create graphic libraries, as well as templates of authoring processes. The WISE environment also includes a debug environment which provides screen identification for the author, a trace file of all "frames" accessed, and access to the state of the variables being used.

The WISE system is not a huge monolithic authoring system, but appears to recognize that the production of different types of courseware requires different sets of tools. It is claimed that new tools can be added to WISE to increase the need to accommodate special courseware authoring needs. Fairweather and O'Neal also recognize the importance of having an efficient debug environment, an environment frequently missing when other authoring systems are being described.

The WISE authoring system clearly reflects the experience of the designers with CAI courseware development over many generations of CAI software systems.

2.2.2 Tutor Based Authoring Systems

Control Data Corporation has developed six authoring systems (Wolman, 1986) each providing a facility for creating a particular type of instructional or testing strategy. The six systems would appear to have a number of common software components, particularly with regards to answer analysis, editing, and the use of preformatted menus. The six systems are as follows: Advanced Tutorial Model, Certification Testing Model, Drill and Practice Model, Graphic Simulation Model, Situation Simulation Model, and the Tutorial Model. Prices reported for long term lease arrangements vary from $3000 (Situation Simulation Model) to $12,000 for the Graphic Simulation model.

Two authoring systems which relate directly to the Tutor language have been developed (Klass, I984, Bagnall and Szabo 1984). The TenCORE system (Klass, 1984) is a software system designed by the team which was involved in the original design of the PLATO system at the University of Illinois. The TenCORE language and authoring system is said to consist "of an easy-to-use authoring language complimented by a closely integrated set of authoring and graphics utilities". The authoring language consists of the 165 Tutor statements described earlier. The graphics utility provides for the use of cursor, light pen, or mouse for creation of graphics, and the graphics can be edited. The code generated is then inserted into the author's Tutor-like program code. A similar editor is available for the creation of special character fonts. Simple key strokes are used to test changes to program code providing a rapid shift from author to execution modes. A student record keeping system and router is available with the system.

The second system called Omnisim (Bagnall & Szabo, 1984) also makes use of the Tutor language, but the interface is provided through a series of templates, some of which are provided as part of the system while others can be designed by the author. Omnisim provides for a separation of subject matter content from lesson logic, and the separate editing of each. The major authoring unit of Omnisim is the display. Each display may contain a number of sub-units or procedures such as text, pictures, erases, pauses, response, etc.

The Omnisim system appears to be more of an authoring system than does the TenCore system based upon available documentation. Omnisim makes provision for the creation and modification of text, question types, answer analysis procedures, and graphics without direct reference to Tutor code. The interface of the processes to the author is made through templates each having various small menus detailing the choices which are available. Through the various menus, text can be edited, and moved on the screen, special character fonts can be used, and line lengths changed. Some templates provide for specific types of questions and answers, for example, true/false, multiple choice, and free response modes. Diagrams can be created, stored, recalled, and edited.

2.2.3 TICCIT

The TICCIT system was probably the first computer based instructional system which had instructional strategies inherent to its software and hardware. The student is provided with special learner control keys identified as rule, example, practice, objective, help, advice, easy, hard, and map. One of the author's function is to provide the content related to all of these special purpose keys, with the exception of the content map which is provided by the system. The system provides a large degree of student control, and indicates to the student by colors in the map the topics which have been covered. Merrill (Merrill,1985) claims that the system uses an expert system to advise the student as to the next best course of action. Merrill also acknowledges that the instructional strategy embodied in TICCIT, rule-example-practice, was designed primarily for teaching classification concepts, procedure-using, and understanding of principles.

2.2.4 The ADAPT Authoring System

A version of the TICCIT system for micro computers, called MicroTICCIT, has been reported by Mudrick (Mudrick, 1984). Mudrick claims that an authoring system is "... an on-line editing package for the creation of courseware", and his report of MicroTICCIT describes 3 levels of editors. He also states that an authoring system should recognize that different authors have different levels of expertise and that different instructional objectives require different instructional strategies. He reports that MicroTICCIT has three levels of authoring format, a) a prompt-driven format, b) a menu-driven format, and c) an author language programming format. The latter is considered to be useful to the author having the greatest amount of expertise and training, and provides the greatest amount of flexibility in the creation of courseware. The beginner's level is prompt-driven with limited capabilities, while the menu level has a simple level and a more complex level. The display editor also reflects three levels of increasing capability, with the highest level allowing the use of specific programming commands. A facility is provided to create basic screen formats, and then to add to these displays more specific features, such as answer checking, video, audio, time limits, etc. To provide for increased efficiency some authoring options may be chained or sequenced

2.2.5 The EASE Authoring System

This is a commercial system which provides both computer managed and computer assisted instruction facilities (Heck and Grimm, 1984). The courseware is created in units called "items" which appear to be equivalent to a "screen" although material may be accumulated to a screen display. Items are defined by simply using an alphabetic character. It is claimed that tutorial, drill, and simulations can be created by the author through the use of menus and tables. Sixteen correct-type answers can be defined, and 8 incorrect types may be defined. Answer analysis can be by keyword or numeric range, and specific feedback can be given to an answer category. A full screen editor is available.

2.2.6 An Apple-based Authoring System

Courseware, Inc., a well known American developer of CAI material, has developed an authoring system for use with the Apple line of computers. Text may be placed directly to the screen, with color and size changes made through use of single keys. No graphic creation is available, but bit graphics created on the Apple by other software may be used and modified. Question formats include multiple choice, true and false, and fill in the blank. Individual reprise statements can be given for each answer type. Screen displays are linked together to form the course, and students are provided with a choice of taking units of course material through a menu system.

2.2.7 CBT+

CBT+ is a micro-based authoring system created at the University of Alberta under contract to Computing Devices of Canada. The product was created to support training of armed forces personnel required to operate radar equipment commissioned by the Navy from Litton Industries. The hardware uses an IBM-type PC and two color monitors, one of which can handle digital and the other video signals, and a Sony video disk player. The system is very modular in design, incorporating menus which give the author access to the creation of testing, tutorial, and animation sequences. It provides extensive support for the instructor in monitoring and controlling the movement of students, and the effectiveness of questions posed to the students. On-line screen creation and graphics are also available. Like PCD3 (developed by Control Data Corporation) a graphical representation using icons to represent various instructional strategies, conveys to the author the structure of the course being created. This system was still being developed in the summer of 1986.

2.2.8 Course of Action

Course of Action developed by Authorware Inc. is a very recently developed authoring system designed to run on the MacIntosh micro computer. Because it is so new, documentation available is still considered to be a temporary version, and a run-time system has yet to be distributed.

Because Course of Action runs on a MacIntosh micro computer, it makes extensive use of the MacIntosh resources, such as pull-down menus, "click-and-drag" mouse operations, and icons. The developers claim that the authoring system is both a courseware design and an authoring system, because it features a "design window" and a "delivery window".

A course is designed by sequencing a set of connected icons. There are 8 different types of icons: display, animation, erase, wait, decision, question, calculation, and map, of which the first 7 represent instructional moves. A course takes on an iconic representation (a map) by connecting together a sequence of icons. Associated with each icon is a set of windows through which parameters are displayed for specification (toolkits).

Graphic procedures used are characteristic of independent graphics software available on the MacIntosh and include shading and animation. Variable data can be formatted on output to the screen. Decision making includes sequentially invoked paths, and random (with and without replacement) path invocation. Input attributes are very flexible and include keyboard, "click/touch", and move an object. Answer analysis procedures are detailed as options in a window and include transformations on capital letters, spaces, extra words, word order, number of words to match, wild character and word definitions. Arithmetic expressions can also be defined using author created variables as well as system and question-related variables.

Course of Action represents a major shift in authoring systems towards the use of the MacIntosh and its software resources, as well as representing a course first with a flow-chart of connected icons and then the definition of the screen display.

2.3 Characteristics Not Described

Unfortunately, few descriptions of authoring systems provide information about characteristics related to their operational or run-time environment. There are important components of an authoring system which are not directly related to the creation and modification of the courseware. These are as follows:

1. provision for the continuity of courseware development from one session to another
2. error checking and the adequacy of the error messages

3. the manipulation of large blocks of courseware
4. error recovery procedures
5. the extent to which the authoring system protects its own integrity
6. the flexibility for creating a wide range of instructional procedures
7. debug procedures, their immediacy and extent of relevant information provided to the author
8. author support systems, e.g., memos, notes, comments, curriculum map, storage of frequently used graphics, text, equations, etc.
9. provisions for backup of files or other units defining courseware objects
10. the interaction between the courseware and the run-time environment including student and instructor support services
11. the definition of a student model and the nature of what characteristics the student model is modeling
12. a conception of what the authoring process is about, its primary objectives, and how these relate to the implementation of instructional strategies believed to be relevant by the author.
13. how much courseware can be created as one unit and be efficiently handled within the authoring system environment

2.4 Summary

Although authoring languages have been improved over the last few years, their characteristics still warrant the development of more efficient and effective vehicles for courseware development. The large number of "elementary" operations available in authoring languages may provide in some instances a finer and more efficient fit between courseware design needs and the final product, however, their ease of use and maintenance costs are generally unacceptable.

Many systems have been described as "authoring systems" for the development of courseware. Many of these appear to be designed for creation of specific types of instructional strategies used in specific instructional environments, and with specific types of computers. Those systems with greater generality are complex systems from a software point of view. The price paid for simplicity of use is usually a restriction on the nature of the instructional strategies which can be created.

CHAPTER 3

SPECIFICATIONS OF AUTHORING SYSTEMS

3.1 The Authoring Process

Although many textbooks describe the formal processes which authors, or teams of support staff could use for the development of courseware the literature has very little to say about the authoring process as it relates to how the author interacts with an authoring system. Perhaps this is due to the predominance of models for courseware development which apply a "systems" approach in which major steps are isolated and assigned to different members of a team without regard to the nature of the computing software systems being used. Such models usually assign distinct roles and functions to the subject matter specialists, instructional designers, writers, programmers, graphic artists, and AV specialists. Such a team approach, although more frequently found in commercial companies and colleges, is seldom available to university staff and teachers in the field. Although the use of an authoring system does not necessarily negate the need for technical support staff, it can place into the hands of the instructor a more direct control of the courseware creation process.

The number of participants contributing to the creation of courseware may vary from one to many depending upon the institutional environment and its financial support of course development activities, and the skills of the contributing members. It is not uncommon for one person, usually the instructor, to take on the complete task of courseware development. Regardless of whether one person, or many are involved, the nature of the authoring software system should be able to provide the required environment; if one person is involved, then a system which integrates all the supporting software is desirable; if there are many contributors, then more direct access by each member to the required tools is desirable.

The paucity of research into the requirements of authoring systems likely stems from the complexity of instruction, as well as from the difficulty in translating those factors supportive of learning into finite instructional sequences. In addition to these problems, the characteristics of students can be highly heterogeneous, requiring at least to some degree, a dynamic adaptation by the instructional system, and all authors do not have the same style in approaching the instructional task.

Merrill (Merrill,1986) has attempted to define six basic requirements of an authoring system. His definitions are useful in the sense that the requirements relate more to learning and instruction rather than to computer language or system capabilities. The six basic requirements defined by Merrill are as follows:

1. definition of instructional transaction archetypes which include presentation procedures and testing/practice procedures. He notes the following examples:
    a. Presentation: text, demonstrate, compare, converse, and explore; explore includes exploration of static (declarative) data bases, and dynamic exploration (simulations).

b. Practice/Test: question, name, classify/decide, edit, calculate, graph/chart, diagram, set dials and knobs, set parameters

2. Instructional Strategies: the authoring system should allow for combining of instructional transactions such as naming, explaining, classifying, deciding, and calculating.
3. Selection of content Structures: this system capability is required in order to organize the subject matter to be taught. These structures may include lists, a taxonomy, and decision structures; they may be static or dynamic.
4. Student management: a system is required which will give advice to the student with regards to the options the system is capable of handling, e.g., a change in the type of instructional transaction, content being studied, changes in content sequences. The authoring system should elicit from the author those parameters required for effective execution of the student management system.
5. Various levels of interaction for the author: an author should be able to draw upon templates, create templates, and use parameter selection menus.
6. Guidance to the author: the authoring system should provide guidance to the author in selection of goals, content structures, useful strategies, and learner management parameters.

In order to handle the complexity of authoring, Merrill also notes that a number of different types of editors (guidance systems) are required to guide the creators of the courseware.

3.2 General Characteristics Of The Authoring Environment

There are a number of general characteristics applicable to any authoring system which were identified by our research. These are as follows:

3.2.1 Flexibility

Authoring software must allow flexibility for the author to create courseware in a manner accommodating a normal or natural style of design. This style may vary with the different authors or team members. It may also vary with the nature of the courseware, such as, for example, the subject matter and the instructional strategies being used. It is unlikely that sufficient flexibility in all aspects of an authoring system can be achieved by one monolithic system. Later we shall define those aspects which are likely to require specialized tailoring.

3.2.2 Minimize Author Memory Load

An authoring system should minimize the memory load and complexities of interaction during the creation process. Implicit in this statement is that the courseware being created is not of a trivial nature and may involve tens to hundreds of hours for a student to complete. In this instance, we distinguish between "lessonware" and "courseware" with the latter being considerably more extensive, and relating to the traditional definitions of a "course". In some instances, the minimization of memory load will depend upon the manner by which the course is represented within the computing system and the support facilities provided.

3.2.3 Adaptation To Level Of Expertise

In most educational settings, authors will have varying degrees of expertise, in some cases trainees, while in other cases experienced authors. An authoring system should have the capability for adjustment to the level of expertise of the user. Such flexibility is likely to promote greater efficiency.

3.2.4 Breadth Of Instructional Strategies

Frequently, developers of authoring systems envisage a restricted approach to instructional design and the instructional environment within which the courseware is to operate. This is evident in those authoring systems which are primarily geared to a particular style of instruction such as, for example, the presentation of test questions, drills, and simulations. Frequently, all such instructional strategies are required within the context of one course, and the authoring system should be able to accommodate such diversity.

3.2.5 Model Of Curriculum Content

Implicit to the design of authoring systems is the model which represents the structure of the course. This structure may in its simplest form be the sequential presentation of course material according to a precise set of conditions implicit in and interwoven with the content. It is desirable to have a modular or structured representation of the content (fixed and generative), the functions controlling the presentation of the curriculum, and a representation of the student, i.e., a student model. In some instances, the curricular aspects may be represented as a more generalized knowledge base structure, although the latter is a goal still to be realized for any useful amount of courseware.

3.2.6 Interaction With Peripheral Processes

Since a course is most frequently operated within the context of some educational program, authoring software must accommodate access to such peripheral functions as student registration, access to other courseware, monitoring student progress, time on system, performance records, and examinations.

3.2.7 Fit To Run-Time Environment

Courseware without the support of a run-time environment serves little more than a demonstration function. It is necessary that interactions between the courseware and the run-time environment be part of the design of the courseware, and therefore, of concern to the author. For example, an author may wish to enable or disable certain student privileges, such as branching, during execution of an examination sequence. The author must be able to identify the specific domain over which such restrictions will apply.

3.2.8 Minimization Of Errors

One of the primary aims of an authoring system is to minimize the chances of errors which may appear during the execution of courseware. The errors most difficult to isolate relate to those processes which are transparent to the author during the creation of courseware. Authoring systems should be designed, as far as possible, to provide adequate representations of those processes which by their nature are transparent during creation, e.g., logical decisions based upon the value of a parameter which is dynamically calculated during run-time.

3.2.9 Fit To The Debug Environment

Since it is unlikely that all errors can be trapped during the courseware creation phase, there must be some interaction between the representation of the courseware and the debugging facilities which are available to the author. Inherent to the debugging environment is the editing environment. All three components, courseware creation processes, debugging processes, and editing processes must be able to interact.

3.2.10 Veridicality Of Authoring Actions

Some actions during the creation phase, such as placing text or graphics to the terminal screen, are perfectly related to what will be presented to the student during run-time. Many other authoring actions will have lesser degrees of veridicality and will require some form of transformation before an adequate representation can be made to the author. For example, invoking a complex answer judging function which may have many steps is likely to require a special representational form for the author to fully comprehend and use.

3.2.11 Retreating From A Process And Error Recovery

The process of authoring a course never goes perfectly, nor always according to a preconceived design. Sometimes an author wishes to "back out" of a process, and thus consideration has to be given in the design of an authoring system to the reversibility of authoring processes. Not all processes lend themselves to easy reversibility, other than at the editing phase. In addition, recovery from critical errors must also be considered. The authoring software system should never "crash".

The degree to which this goal can be reached may to some extent be contingent upon the environment under which the authoring system operates, the programming language used in its creation, and the characteristics of the computing system being used. The authoring system must always be sensitive to protecting its own integrity regardless of the actions of the author.

3.2.12 Help Systems

Software systems which contain provisions for the user to request help at almost any stage of defining a process can reduce errors and frustrations. In some instances the need for help can be prevented by careful consideration of prompts and monitoring for logical inconsistencies of action by the author. Help is likely required where terminology, as might appear in a menu or an error message, is not meaningful to the author. In some instances "embedded training" (Kersley,1985) may be useful in guiding the author through the definition of complex processes. Careful selection of commands, and their use in a consistent manner is helpful. For example, the commands exit, quit, stop should be used in a consistent manner throughout the authoring system.

3.2.13 Windows And Menus

In general the use of windows which contain menus is a useful procedure for presenting current lists of options to the author. However, if many menus are available considerable inefficiency can be introduced especially when the author is not familiar with which menu contains the sought after action. In addition the depth to which one window may call another window must be considered in the context of the memory load placed upon the author. Menus may take on a number of different forms, in some instances they may display iconic representations of an action or process, command words, or information required by an author. If windows overlay screen displays being created by an author, then a new dimension of software problems are introduced in the management of such windows and the reconstruction of appropriate areas of the overlayed portion of the screen being created by the author. Windows can be an ideal vehicle for presentation of graphics with differing grid resolutions if the capability is available for assignment of the window coordinate system. Windows can also be useful during authoring, debugging, and execution phases to show the results of different processes.

3.2.14 Special Terminal Functions

The use of "pf" keys can be an efficient process by which authoring actions can be initiated with a single keystroke. Errors due to incorrect command specifications can be reduced. However, keystroke errors may result in processes which are entirely out of the context of the current process. In such cases the keystrokes should be trapped with an appropriate error message or a request for confirmation. A definition of the purpose of each function key should be available to the author.

3.2.15 Courseware Optimization

Seldom are the initial passes at courseware development and testing indicative of completely acceptable performance. The ease with which courseware can be optimized depends upon a number of authoring procedures, including the monitoring and editing software. Optimization may include deleting old content, adding new content, changing answer analysis procedures, etc.

3.2.16 Formal Testing Systems

There is a special need in some educational environments for the author to be able to easily develop formal testing procedures in addition to those used for purposes of diagnostic and monitoring intermixed with courseware presentations. There are a large variety of testing formats which can be used in the presentation of questions, as well as the manner by which the author selects questions for presentation. In the most complex case questions may be selected on the basis of Item Response Theory or their diagnostic power. Considerable complexity can also be introduced when a simulation is required, such as in patient management examinations used in various fields of medicine. The latter become complex because frequently many bodily systems have to be "run in real time", and be continuously available for assessment at the option of the examinee.

3.2.17 Access To External Utilities

In some subject matter areas, access to external support utilities are required by the author as well as by the student. Glossaries, numerical tables, and specialized mathematical routines may be required for effective courseware presentation.

3.2.18 The Editing Process

The editing process is a crucial component of any authoring system. In fact, in some respects the authoring system can be described as a specialized editing system because the editing actions imply the creation of courseware as part of the modification processes. Complexities are introduced by editing visible complex screen displays, but are increased significantly when editing those components of the courseware which do not have visible screen representations. Visible screen representations would include graphics as well as text. However, other processes such as response judging do not usually have obvious screen representations. In addition, the editing of graphics can be a complex process depending upon the nature of the data structures holding their representation. Further complexities are introduced in this regard if consideration has to be given to the transmission of graphic information from a host to a terminal. Depending upon the nature of the data structures holding the course representation, immediate assessment of the editing process can be complicated when branching is encountered.

3.2.19 "What You See Is What You Get"

The capability of an author to see directly what is being created is useful particularly for creating the visual components of screen displays. This is especially useful when textual and graphical material is being created, as corrections can be made easily and the author can judge the esthetic appearance of the screen. There are some situations where this may be cumbersome, as when the amount of material to be displayed is dynamically controlled and must fit a prescribed restriction. In the latter case dynamic formatting is likely to be desired. In some instances, indirect formatting is very much akin to the functions of text processors, which may be desirable from an efficiency point of view, but may encourage the author to create undesirable lengthy sequences of text for the student to read.

3.2.20 Providing For Continuity

Not only do some authoring systems assume that lessonware rather than courseware is the desired objective, they also assume that the author will create a complete instructional unit in on sitting. Although the latter may be true in some work environments, if the author is also an instructor or teacher, the case is more likely that several hundred sessions may be required and support services are required to maintain continuity. In addition, consideration needs to be given to the situation in which an author may be working on more than one course over a lengthy period of time.

Continuity can be provided by the authoring system storing author notes, dates and times, as well as course characteristics such as the number of units currently developed. Such information will also increase the confidence in the author that the desired course section has been entered.

3.2.21 Backup Services

Backup services are generally assumed on large mainframe computing systems. However, increasingly micro computer users are becoming aware that backup is their own responsibility. Backup services may be considered fixed and automatic, or as a option to the author upon entering or leaving the authoring mode.

3.3 The Generality Of An Authoring System

When all the general points noted above are considered, and to these are added the variability expected in author competencies and subject matter characteristics, the question must be raised as to whether one general authoring system can be created to serve the needs of all authors.

3.4 Software Maintenance

Schneidewind (Schneidewind, 1987)) has described the general problems associated with software maintenance. An authoring system is likely to be a complex system of inter-connected modules which invariably will require maintenance as bugs are found and modifications and enhancements required. Unfortunately, much software is not designed with maintenance in mind.

The design of an authoring system should take into consideration maintenance problems, such as for example the isolation of bugs. For example, our experience suggests that it is impossible for an author, experienced or otherwise, to accurately describe the actions prior to a system malfunction. This suggests that, as an option, all keyboard responses should be recorded. In our preliminary trials of one version of the authoring system used by students, such recordings proved invaluable in isolating problems, and confirmed the fact that error reports given by the user were usually quite inaccurate.

Our experience suggests that one of the most difficult problems to isolate has been described by Schneidewind as the "ripple" effect. A modification to one process, may cause a cascading effect throughout many other processes. Frequently this effect is not anticipated unless a complete and accurate documentation of each process is available. Ideally, specialized software capable of determining the extent of the "ripple" effect is most desirable.

3.5 Human Factors in Interface Design

With the advent of micro computers and their many graphical and animation software resources there is a tendency of some designers to view the use of these resources as a challenge in combinatorics. A number of researchers (Morlund, 1983; Foley, 1984; Bourneque and Treu, 1985; Houghton, 1984; McCann, 1983; Murch, 1984; Reilly and Roach, 1986) have recognized the problem of poor interface design and provide many worthwhile guidelines.

Morland (Morland, 1983) notes two major reasons for too much complexity in software interfaces:

> "The first is the typical programmer's fascination with intricate mechanisms. There is a widespread tendency for programmers, even good programmers, to embellish their work to the point that clean, classical architectures are transformed into baroque monstrosities. .....
>
> The second and most prevalent source of complexity is the distribution of interface design and implementation responsibility among several people."

Morlund notes that interface control and standardization should be an early consideration in the design process in order to:

> "1) provide a unified and consistent interface;
> 2) maintain a focus on operational simplicity and clarity;
> 3) develop a user-friendly approach to error handling;
> 4) assure that user interests are properly represented when design trade-offs are evaluated."

Although good documentation describing how a system operates is important, it does not replace good human interace design. It is prevalent in some environments, particularly those which can afford a large number of specialized human resources, to have technical writers responsible for documentation. We concur with Morlund when he states ir this regard:

> Writing documentation is (an) anathema to most programmers, and so many manuals are written by technical writers who have only a scant knowledge of a particular system. This is a perfect illustration of what is known in some circles as a 'Boyce Compromise' -- a solution that cleverly enables all parties to lose. Either a progrmmer is forced to write about something he understands but cannot describe, or a writer is forced to write about something he would describe if he could understand it".

Morlund makes the following suggestions with regard to video screen layout designs which are relevant to the interface design for an authoring system:

1) "Simplify, simplify, simplify!"
2) Elimination of social amenities; this does not imply that interactions should be uncivil or impersonal.
3) Do not "talk down" to the user; attempts to be "chummy" with pseudo-personal greetings" should be avoided.
4) Use screen titles; have a one-to-one correspondence between menu items and subsequent screen titles.
5) Use special screen displays for advanced users which will allow for more compact displays and abbreviated responses.
6) A hierarchical level of error messages which is invoked with greater explanation when the same error reoccurs.
7) Standardize screen terminology, e.g., enter, input, give and type are likely to indicate the same action to the user.
8) Color can be used to link related information, differentiating optional features, highlighting errors, and separating prompts from other information.
9) Special function keys can be useful, but "they can be confusing for novice users and restrict future systems developers and should be used advisedly".
10) In order to prevent crucial errors, verification should be requested.
11) Notification of an error should be made contiguous in time and space to its commission.
12) On-line systems should have an error logging capability.

An important consideration in the design of an authoring system is the cognitive model which the user is likely to use in understanding the domain of interactions. This model may be made explicit through documentation, but it is likely that it is created by the author on the basis of interacting with the authoring sytem. This creation should have a high degree of veridicality with how the system actually operates. Foley, Wallace, and Chan (Foley, Wallance, and Chan, 1984) suggest that this is the first step in the design of a system, followed by defining detailed semantics, and then the language syntax. Their definition of the syntax is very much like the basic syntax of Elf. (See Appendix C).

These authors also suggest that the primary criteria for determining ergonomic quality are:

a) the time any user must spend to carry out a task
b) the accuracy with which this can be accomplished, and
c) the pleasure the user derives from the process.

They note further that:

> "In a pure-production, low-creativity environment, the third criterion may not be judged important, while in a creative, voluntary environment, maximization of that factor may be the dominant goal."

Clearly, the authoring environment is, or a least should be, highly creative for most instructional tasks. Secondary ergonomic criteria, some of which have already been noted include the following:

1) learning time
2) recall time
3) short and long term memory load
4) error succeptibility
5) fatigue suceptibility
6) naturalness, and boundedness (convenience)

Boundedness is of particular importance in the design of an authoring system. Foley, Wallace and Chan have defined boundedness as "the size of the space in which one must work -perceptually, cognitively, or mechanically". Cognitive boundedness is of particular importance as an author must maintain contact with the concepts to be taught, how they are to be taught, and at the same time interact with the complexities of the authoring system to create the desired features and flow of the course.

CHAPTER 4

THE STUDENT MODEL PROBLEM

The purpose of defining a student model as part of the authoring system is to optimize the presentation of the curriculum and the application of learning strategies to maximize the learning of the student. How learning may be maximized depends very much of how one perceives a student learns, and what factors are believed to influence learning. How one perceives the dynamics of learning and the variables involved depends very much upon one's theoretical position as to the importance of such factors as maturation, past experience, ability variables (e.g., spatial, numerical abilities), intelligence, anxiety, personality, motivation, and perception just to mention a few. In addition specific variables are frequently identified when specific learning dysfunctions exist. These latter variables may be more or less important depending upon the subject matter being taught.

In reviewing the literature on student models as it relates to computer-based education, the following points appear to be pertinent:

1. The use of variables to hold information about the student, usually current information generated by the CAI program and expressing some measure of progress, is characteristic of early CAI efforts. The use of internal counters, strings, and switches for this information was usually not identified as being a "student model" but processes of "score keeping".

2. The more recent literature, usually in the area of intelligent CAI or expert CAI, defines a student model as it relates to the schema of the content being taught and the extent to which information about the student collected during execution of the CAI course approximates this schema. Thus, "semantic networks" inherent in the subject matter are related to the "semantic network" inferred by the student's actions in the course. Others have suggested that a formal semantic be used to describe the schema of a course, and the progress a student makes is interpreted relative to the semantic description.

3. Some authors use the term "student model" but are so vague about the nature of its content that little is gained from their definition; others may define a student model with one or two parameters. (O'Shea et al, 1984)

4. Authors in the area of intelligent CAI tend to view a "student model" as a program characteristic which is able to identify or diagnose errors made by the student, and to identify and correct such errors within a tutorial interaction with the student. (It would seem that some authors in this area think that every student is of above average intelligence, will automatically learn when an incorrect deduction or induction is identified and a correct approach suggested or provided, has an almost perfect memory, has no motivational deficits, can read with perfect comprehension, etc., i.e., the whole area of learning theory is ignored.) However, an interesting concept suggested by the AI oriented researchers, is that a schema of a course (its basic concepts and interrelationships among them) could be

represented by some formal semantic, and that the student's understanding can also be represented with the same formal semantics (Self, 1974). (This position is taken because the same formal semantics in fact define and drive the presentation of the course to the student). The problem then is to determine whether the two representations are equivalent.

Indeed, it would seem that the current literature suggests the story of the "blind men and the elephant" as each researcher sees the student in an idealized manner with little regard for the students with whom the majority of teachers and instructors must deal with on a daily basis.

4.1 A Broad Student Model

The student model should be a representation of the student not only in the context of the CAI instruction, but also in the context of the past, and perhaps even the future. Information about the student is required from the past and outside of the CAI instruction (external), as well as from the processes inherent to the CAI instruction (internal).

4.2 Examples of External Information

Information external to the CAI instruction which could be relevant to optimizing the student's path through the content and the strategies used are as follows:

1. Achievement information: For example, last year's mark in arithmetic, spelling, reading; has the student had calculus, matrix algebra, biology xxx, last year's average; pre-test achievement level

2. Psychological information: anxiety in general with reference to instruction, specific to testing taking, specific to CAI; attitude towards the subject being taught; maturation and age; Piagetian stage of development; ability to generalize and make inferences, learning "style"

3. Special dysfunctions: poor reading ability with difficulty in figure-ground separation, reversals, eye-hand coordination, deafness, visual handicaps, English as a second language

4. Abilities: spatial, numerical, verbal IQ, general IQ, short term and long term memory

5. Other: previous use of the CAI system

None of the literature reviewed considered a student model in terms of the variables noted above. One reason might be that some of these factors would be inherent in the design of the CAI course, eg., a remedial lesson for students who cannot add two fractions would not need to hold information that the student cannot add two fractions.

4.3 Examples of Internal Information

By internal information we mean that information which is collected by the system as the student interacts with the CAI course. (In the case of a computer administered pre-test, the interaction is current and within the course, but essentially one is attempting to ascertain past experience). Typically, information in this category has been a count of the number of questions answered correctly, incorrectly, latency time, etc. These are useful, but not very helpful in terms of accurately describing a student's characteristics. An accurate description may well include latency time, but should also include the type of information which can be obtained or at least inferred by a diagnostic approach to the answers given by the student. This suggests that the answer analysis component of a CAI system must provide more than right, wrong, and unrecognized categories.

A critical issue is how to determine the extent to which goals of the instructional processes are being attained. The level of achievement attained needs to be contrasted with the level of attainment desired, i.e., how close is the fit between the instructor's goals and those actually reached by the student or group of students.

Obviously, the capturing of internal information must be planned for in the design of the course and within the environment of the system executing the course. For example, the system must know precisely where an item of information is generated (eg., the student is in addition of 2 digit numbers), and what kind of error if any occurred.

The AI oriented researchers appear to implicitly considered the reliability of a student's answer as being almost perfect, while others consider a student's answer as having a random error component and thus requiring many more exemplars before an inference can be made about the state of the student's knowledge or processes (Tatsuoka, 1987). The AI researchers also place importance on having a natural language parser for analysis of student responses. However, no one reports the effective and efficient use of a natural language parser in the context of non-trivial courseware. In addition some researchers assume that a perfect correspondence exists between the student's behavior (response) and the presence or absence of understanding.

4.4 The Purpose of a Student Model In CBE

The student model is taken to be a set of statements and variable attributes which characterizes a student with respect to an instructional task. This information, or transformations of this information, are used to control the instructional processes, and to optimize the attainment of the instructor's objectives. Thus, the nature of the student model is such that it may reflect the author's theoretical position on the importance of a wide variety of factors which affect learning as well as the nature of the instructional processes. For example, an author who views the learning of a list of foreign words as a conditioning paradigm, implicitly defines a position on the learning process of the student and the instructional processes of the CAI program. In this sense, a student model cannot exist in isolation of the instructional processes and the content of the courseware.

What should the student model do? The student model should:

1. describe the characteristics of the student prior to taking the course
2. describe what the student has learned in terms of specific topics and concepts
3. describe the level of achievement
4. gain new information and insights about the student's characteristics and use this information to optimize learning for the student by adjustments of course parameters, such as supplemental sequences, answer diagnosis, etc.,

A major problem is how to capture this information from the author. The author may not be able to specify the nature and interrelationships of the information in one grand design. It would seem the task will require a computer supported approach.

The student model proposed here requires the derivation of external and internal information and specifications of how this information is used by the control component.

4.5 The Use of External Information

Information about the student external to the system such as reading comprehension, IQ, previous achievement, can be handled by the definition of appropriate variables. These would have to be defined as to type, and made available at a global level so that they could be used throughout the run-time system. A method for defining and reviewing these variables by the author is required. A pre-test, although collected internally is to be considered as a measure in the class of external information.

External information about students becomes useful to the author if that information is sufficiently related to learning outcomes that it can be used to parameterize or modify the presentation of the courseware. The courseware must be built in such a manner as to accommodate the application of such variables. Unfortunately, if this relationship between external information and course presentation is considered as an application of aptitude by treatment interaction, there is not an abundance of research to support its effectiveness. There may be a number of reasons for this including the following:

1) The treatments, although different are static and non-adaptive to the use of external information.

2) Effectiveness has been usually summative, which may be affected by many other intervening variables, rather than assessed at many intermediate points.

3) The variables may be important only at specific intervals of the instruction.

4) The treatment is sufficiently compensatory, or the student reaction to the instruction is sufficiently compensatory as to eliminate interaction effects.

Assuming that the author believes that external information is relevant to the modulation of the course presentation, then the authoring system must allow for its formal specification, or alternatively the system must have a sufficient degree of expertness to learn such specifications (Nesbit, 1988). If external information is not generated through such methods as pre-testing, than the run-time system must accommodate its entrance into the courseware, perhaps as part of the student's registration.

4.6 The Use of Internal Information

Assuming a reasonable degree of interactiveness in the courseware, a considerable amount of information can potentially be gained about the student, which in combination with the external information could be used to modify the presentation of the course in order to optimize learning. Internal information may be relatively subject matter independent such as latency time, or subject matter dependent such as a specific question being answered correctly or incorrectly. The interaction of external and internal information and its application is illustrated in Figure 4.1.

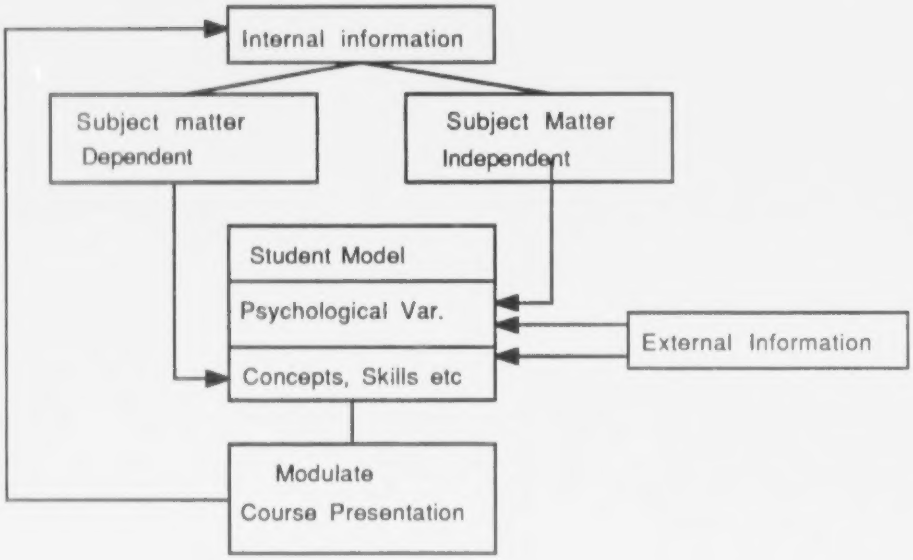

Internal information
Subject matter Dependent
Subject Matter Independent
Student Model
Psychological Var.
Concepts, Skills etc
External Information
Modulate Course Presentation

Figure 4.1 Application of External and Internal Information in the Student Model

Figure 4.1 suggests that somehow the psychological part of the student model, the part representing concepts and skills specific to the subject matter, and external information must be transformed in such a manner as to modulate the presentation of the courseware. Specifying the transformations and applying the result to courseware presentation is likely more an art of the author based on experience, than a science. In addition the concepts and skills being learned by the student must be matched against the final desired set of concepts and skills in order to assess dysfunctions and to determine when the instruction has been completed. A diagnosis of specific dysfunctions may have to be related to remedial components of the courseware.

The application of a student model has a number of factors which must be considered:

1) The validity and reliability of external information may vary initially and during instruction.

2) The apparent validity of internal information is likely to be dependent upon the author's experience and even the method of instruction.

3) The reliability of internal information may be unknown, but certainly not perfect especially when dependent upon a single response of the student.

4) Internal, subject dependent information, must be related to the concepts and skills which are to be learned. This implies that questions in the courseware must be "tagged" as to the kind of evidence they are intended (or believed) to provide.

If concepts and skills are identified in a taxonomy of some type, then the result of analyzing the student's response, and perhaps even the inference which can likely be made from the response, must also relate to the cells of the taxonomy. Parenthetically, one must keep in mind that NOT all questions asked of the student will be related directly to learning objectives. For example, one or more questions may be asked to ensure that the student is examining the appropriate information on the screen. This implies discriminating among the questions posed.

4.7 Analysis Of Information

One or more operators, or processes are required so that the author can determine the status of the student learning. Two levels of analysis are required, one related to specific cells of the taxonomy, and another related to various levels of summarization. For example, a summarization by topic might be required over many levels of objectives. It is at this analysis level that the author may wish to invoke some "expertness" to make valid inferences about the student's progress and where the student should be taken next.

4.8 The Problem of What Should be Modelled

The term student model has a variety of interpretations. For a teacher in a public school, the cumulative record may represent a student model; for a clinician working in a reading clinic the student model may be a set of test scores on various reading tests; for a cognitive psychologist the student model is likely to be a collection of hypothetical components of information processing and their interrelationships; for a subject matter expert the student model is likely to be a description of what the student knows and does not know. Some student models may be represented by a collection of quantitative measures, while other student models may be quite qualitative (Clancey, 1986).

Quantitative student models represented by parameters from external or internal sources are meaningful only when interpreted by someone who can relate these parameters to student behavior. The expertness of the interpretation rests outside of the CBE context although the results of such expertness may be built into the logic of the courseware. It is likely that the expert interpreter will base interpretations on a model of the student which includes affective as well as cognitive components and include some subjective confidence boundaries based on the reliability and validity of the parameters.

However, it is possible to consider the expertness resident within the CBE context. Such expertness may take the form of a qualitative model which describes the appropriate relationships among the concepts being taught as well as processes for inferring from the student's responses, the extent to which these relationships exist within the student. Such models tend to be subject matter specific and may contain specialized software for inferring misconceptions from a student's response. In this latter case the qualitative student model is more a dynamic modeling or emulation of how the student operates within a specific subject matter domain. The complexity of such emulations seems directly related to the generality of the modeling. Models which are based upon a definition of the relationships among concepts and which are used to drive the presentation of the subject matter are appealing when the subject matter is well bounded. The building of models to adequately represent the dynamics of a tutor, the student, and the subject matter is no minor task and require considerably more research before they can be applied in non-trivial courseware.

CHAPTER 5

THE AUTHOR-SYSTEM INTERFACE

5.1 Introduction

Our work suggests that a large number of procedures can be identified which together provide the characteristics of the author-terminal interface. The primary goals for providing an appropriate terminal interface are to allow the author to concentrate on courseware development, minimize errors, and to optimize the efficiency of debugging, making enhancements, and maintaining the courseware.

A number of procedures and techniques can be employed to effect these goals. These are discussed below.

5.2 Level Of Prescriptiveness

Some have argued that the greatest degree of authoring flexibility and power exist when a programming language is used (Merrill, 1985) since generally programming languages allow for very detailed specifications which can be sequenced to represent a wide variety of processes. Of course, the recognized disadvantage of using a programming language relate to the high costs of the human resources required. Authoring systems which are specifically designed to produce a particular type of instruction may be easier to use, but impose restrictions upon the creativeness of the author.

The greatest degree of prescriptiveness is evident in authoring systems which are designed to efficiently produce courseware of a particular type, such as for example, question and answer sequences. Frequently, in such systems instructional strategies are fixed through the use of templates developed using a programming language. The author is usually constrained to completing a form detailing the variable components of the presentation to be displayed to the student.

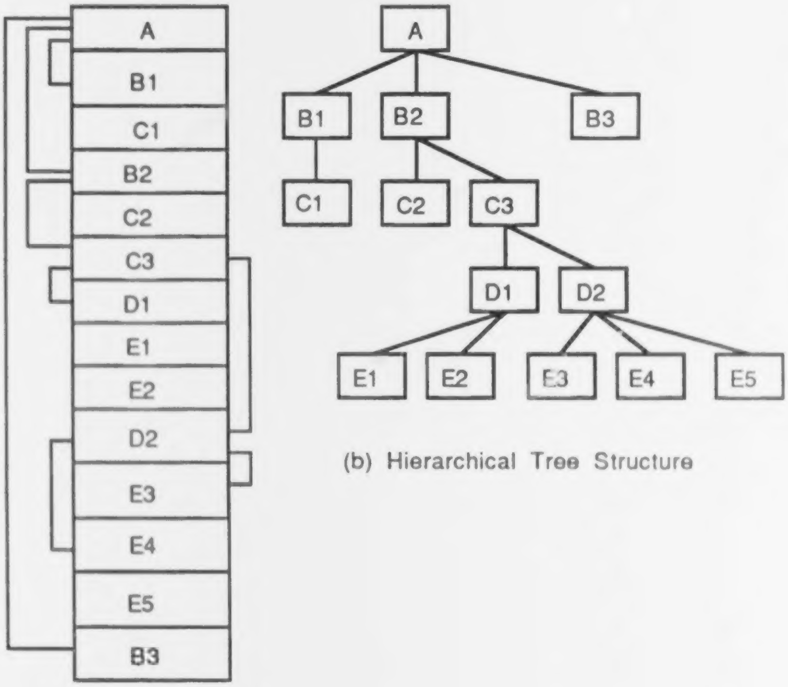
Ideally, a facility which can be invoked to provide different degrees of prescriptiveness is required. During the creation of a test item bank, and the presentation of items in standard format, a high degree of consistency is usually desirable. However, there are other situations in which variability is desired by an author. Not all pre-designed templates will satisfy the more creative authors, who will wish to design their own templates specific to their instructional styles and approaches to their subject matter. Our research suggests that the latter can be relatively easily accommodated into an authoring system, stored in a library of author created templates, as well as providing for easy modification by the author to produce slightly different variations.

5.3 The Course Structure Problem

The presentation of courseware to a student is generally observed to be linear. However, the internal (internal to the computer) structure may not be linear at all. Coursewriter is a good example in which the internal representation of the course is linear. It is linear also in the sense that course program codes follow sequentially, line by line, with deviations in this sequence made only by the use of some type of branching instruction. Branching instructions may cause execution of program code at the beginning of the sequence, at the end, or at any line within the sequence. Depending upon the experience of the programmer, and the number of "patches" placed into the program code, tracking the exact presentation of material to the student can become a horrendous task. The degree of complexity is also markedly increased when the author wishes to manipulate large sections of a course.

The internal structural representation of the course can also markedly affect the ease with which other processes can act upon the internal representation of the course. Specifically, the capability of executing a course sequence, directly editing the sequence, and then re-executing becomes a complex problem. In the worst case poor internal representation of a course prohibits the execution and immediate editing of the course.

Two representations of course structures are diagrammed below in Figure 5.1



(b) Hierarchical Tree Structure

(a) Linear Structure

Figure 5.1 Linear and Tree Structure Representations

The lines representing branches in the linear sequence illustrate the complexity of the paths over which a student may be taken. In the hierarchical organization, paths are clearly defined, and the path to a particular component is almost always known. In addition, in reaching a particular component, previous parameters which may determine the behavior of the goal component can be determined or defaulted. This, in general, is impossible with a linear organization.

As we shall see, how the course is structured internally, will have a marked effect upon the ease with which support facilities can be provided to the author.

5.4 Windows

True windowing provides the authoring process with the capability of making very efficient use of a limited display surface. Windows may be used to provide transient information to the author in the context of a current procedure. In addition, the use of multiple windows can allow the author to track more than one process simultaneously as might be required during debug procedures.

The capability of using windows varies with the type of software and terminal equipment available to the author. The creation of windows, their positioning (tiling) on the screen, clipping (as may be required when only a portion of what is being viewed is actually displayed), and the linking together of windows in a tree structure, can be a complex software problem. If a general system of windowing is available then it can be useful during authoring and execution of a course.

Ideally, windows should be able to hold both text and graphics, and to have information within them scrolled. In addition, it is desirable for a number of reasons one of which is the potential for portability of courseware, to have windows for which different coordinate systems may be defined. For example, it is useful to have a coordinate system origin in the upper left for text, but in the lower left for numerical plotting in the first quadrant.

Since the author should have freedom to use any area of the display screen, windows should be able to be repositioned as the need arises.

If windows cannot be repositioned, our experience suggests that they should be on the right side and at the bottom of the screen. The author can control the extent to which he encroaches upon the right and bottom of the screen. However, provisions are required to easily rewrite the windows and their contents if they are damaged by the actions of the author.

Windows are useful for the display of messages, menus, reference information, display of executed results, and details of the steps in answer judging.

5.5 Menus

Menus may contain named processes or iconic representations of processes. Menus may be displayed continuously, or upon call. The processes represented in menus may, or may not be context dependent.

When menus contain named processes, consideration should be given to making the availability of the processes context dependent, that is, only those processes which are currently applicable should be available for selection. However, some problems can be created by this approach if blindly used. If an instructional exchange with a student is being created for the first time, it is reasonable to restrict the sequence of the processes to asking a question, getting a response, analyzing the response, and effecting a decision. If parts of the same sequence are being modified, then the context of the modification suggests that the author must be able to independently access anyone one of these processes. If the courseware is appropriately structured, the context can be known by the system; if inappropriately structured, the system will not know the context and authoring errors could result.

When processes are named, then consideration must be given to maintaining consistency in the use of the names. For example, stop, quit, exit, should consistently invoke similar procedures, otherwise errors are likely to result in their use.

Menus may be displayed statically or dynamically. Where frequent reference is required by the author to the contents of the menu, then a static display is to be preferred. However, there may be a need to allow the author to relocate this display so that it does not cover a portion of the screen of current interest to the author. (The problem of displaying menus, the tiling problem, has been well presented by Gait, (Gait, 1986)). Static menus also provide the added advantage of keeping all processes in front of the author minimizing the search of menus for a particular process.

Menus may have fixed or dynamically displayed contents. When the contents are dynamic, further problems in software design are encountered since formatting, scrolling and paging of the contents may be required.

5.6 Icons

Iconic representations are currently in vogue as a means for interfacing a user to processes and information. Some micro computers, for example the MacIntosh, allow the author to select the mode of representation, i.e., iconic, or textual. Considerably more information may be contained within textual information although the sequenced space required for display may be a disadvantage at some times. In addition, there are some processes, and sequences of processes which cannot be adequately represented in an iconic form. For example, representing a sequence of steps in parsing a student's input, is not easily represented in an iconic form. Such processes and sequences may be better represented as combinations of icons and flow charts (Chang, 1987).

5.7 Flow Charts

Flow charts generally include some iconic representation of processes. Traditionally, the icons have been simple geometric figures representing computing processes. However, there is a need for an author to be given representations for a wide variety of information, some of which may be represented as flow charts, but others more as tree structures which illustrated how information is linked. The adequacy of a representation and author needs varies according to the experience and background of the author.

The flow charts required in order to accommodate the various levels of author expertise should provide for different levels of detail, each successive level providing more detailed and in some cases complex information. For example, in representing the content of a course, the first level might be a single icon containing the name of the course; a second level when "opened" would show a more complex arrangement of the major units of the course. Opening up one of the course unit icons, would show a representation of the specific topics covered by the unit. A similar representation providing more detail with increasing depth into the icon, is required for the logical processes embedded in the course. For example, at the simplest or highest level of representation, an icon representing an answer checking processes might be indicated; opening up this icon would show a representation of a number of different possible answers which would be checked, opening up an icon at this level would show the detailed logic and matching processes which are involved for one of the possible answer checking paths. Such a procedure requires careful design of the data structures representing the course, as these structures and their informational content have to be transformed into a graphic representation for the author.

For an experienced author, the sequence of operations necessary to get to a desired level of representation becomes time consuming. The system should save sufficient information during the process of getting to the desired level, to later allow a single keystroke to invoke the traversing steps necessary to get to a previously established level.

Because the amount of information which may be available to the author is considerable and display space limited, careful consideration must be given to how information is represented. Where special characters become icons, consideration has to be given to increased memory load placed on the author and increased chances of errors due to a confusion in the meaning of the characters.

5.8 Informing And Helping The Author

Three types of informing actions are required of the authoring system: a) support information designed primarily to reduce the short and long term memory load placed on the author (Foley et al, 1984), b) confirming information to provide the author with information that a process was adequately completed, and c) review information primarily of a historical nature. We distinguish these types from error messages provided to correct an inappropriate action on the part of the author.

In suggesting these 3 types of informing actions, we have in mind courseware of a lengthy nature, e.g., ten or more hours of instruction, and not small sequences of lessonware which may take a hour or less for a student to complete.

Supporting information provided to the author is required where lists of items are expected to be recalled as the author works on the courseware. A good example is a list of diagrams and their associated notes, particularly if the authoring system is creating a library of such diagrams. The sequencing of this information may have to be varied. For example, a sequence according to when a diagram was created is likely to provide a topical context for the drawing. Ordering by name or ID, content, and date of creation are other variations which are useful. The problem of providing information according to different aggregations becomes more important as the length of a course increases.

Where objects are to be identified, such as drawings, the usual tendency is to want to identify these by "meaningful" names rather than some alphanumeric identification. "Meaningful names" soon become difficult to create if a large number are required. For example, at the University of Alberta one course has approximately 350 diagrams which reside as graphics code. Clearly, 350 unique and relatively short names cannot be attached to this many drawings without causing confusion. Some other system of identifying the diagrams is required. It is assumed that such drawings may be called upon many times in different parts of the same course, or by another author for use in a different course.

Confirming information is required specifically when an action does not make an observable screen change during the process of authoring. In many cases a simple feedback such as "done" is sufficiently confirming to maintain a level of confidence in the author that the system has carried out the desired action.

Review or historical information is required to provide continuity from one authoring session to another, and in some cases for isolating system malfunctions. For example, it is desirable for the author to have the ability to leave notes at the end of one authoring session which are then available for recall at the next session.

5.9 Warning and Error Messages

One might argue that ideally, warning and error messages should not be required as the authoring system should be sufficiently guiding to prevent their need. From a practical point of view, messages will be required since even if author actions are all appropriate, system failures can occur. For example, the finite limit of a list structure might be nearing its maximum, a hardware failure may occur, or an operating system error outside the domain of the authoring system may occur.

Messages should be provided as contiguous to the point at which the error was made as is possible. This cannot always be done since a sequence of n actions may be appropriate, but in the context of these n actions the n+1 action is inappropriate. If possible, the nature of the error should be identified and a modified action allowed. Warning messages may be usefully employed as a series of n actions begins to suggest sufficient constraints are being created that an error is likely to occur.

It is likely true that if an authoring system is sufficiently constrained in terms of actions available to the author, the less likely the need for warning and error messages. A template system allowing the creation of a single defined instructional interaction is likely to require fewer warning and error messages than a large and very flexible authoring system.

The flexibility of providing error messages can depend upon the nature of the programming language in which the authoring system has been written. If a highly structured language is used where various processes are represented by tree structures a "cascade" of error messages may result (Efe, 1987). This effect should be avoided.

5.10 Author Entry Mode

At least five different methods of entering information by the author can be defined:

1. keyboard, which may include special keys
2. cross-hairs
3. a "mouse"
4. a bit pad
5. a light pen

Each method has advantages and disadvantages particularly if a wide range of author activities are required. These modes of input may be required not only for entry of course content but also for communicating with the authoring system itself.

5.10.1 Keyboard Entry

In the case of tutorial courses, a large amount of text must be entered by the author, and clearly, the keyboard is the preferred mode of input. The use of optical readers, to read text which already exists, such as in a set of student notes, may increase efficiency of input but decrease instructional quality. Although some popular desk top, micro-based systems provide an excellent variety of character fonts, access to different styles of character fonts is far less important to the author (and student) than access to specialized symbols, and the ease with which they can be positioned on the screen.

Keyboard entry for selection of choices can be controlled, i.e., the cursor can be positioned at the next possible choice, however, if the list of choices is lengthy then inefficiencies are introduced. Keyboard entry has the disadvantage that typing errors can be introduced. If commands are being issued by the author, the authoring system can be designed to accommodate some of these errors immediately. However, this is not the case when content is being entered, although a separate process may be initiated to check for spelling and typographical errors.

The use of the keyboard for entry of authoring commands can be used efficiently by experienced authors, particularly if a "type ahead buffer" is available. However, such buffering of commands can create problems for the naive author because the system response is not clearly contiguous in time to the entry of the command.

5.10.2 Pointing Responses

Pointing responses can be carried out using a keyboard, for example, positioning the cursor to particular spot on the display screen, however this is an inefficient method for entry of information, and provides for a resolution which is usually equal to one character position. If positions to be pointed at are all known, then inefficiencies can be reduced by allowing the cursor to move only to those designated positions.

The use of cross-hairs controlled by keystrokes or by holding down a key, is useful in some graphic modes. For example, a large cross-hair on the display screen can be positioned and alignment with another point quickly verified. Although it may be possible for the authoring system software to position the cursor in the domain within which a point is acceptable, the positioning is likely to take more time than the use of a pointing device such as a "mouse". In systems where the incremental movements of the cross-hair can be controlled by software, there is the possibility of more accuracy, if required, in positioning. Such accuracy may only be required for the creation of graphics.

The use of a "mouse" is fast and well suited to making selections, and for some graphics work. Depending upon the screen resolution, fine positioning can be a problem. This problem is sometimes overcome by software which magnifies the area in which accuracy is required, e.g., magnification of the screen to clearly show individual pixel positions. Software, such as in CBT+, which highlights or provides a color change when the target area is pointed to, but not yet selected, is useful in reducing pointing errors. The implementation of such features may depend, however, on the terminal equipment being used.

5.10.3 Bit Pad

Where irregular line segments are required, such as in a diagram for biology, then a bit pad is likely to be the most efficient manner for entry. In this instances slight irregularities due to hand sketching may add to the authentic appearance of a drawing. However, where a considerable regularity is involved, eg., a sketch of a mathematical functions such as the normal curve, the bit pad becomes ineffective.

CHAPTER 6

AUTHORING THE SCREEN: TEXT

6.1 Introduction

In some instructional strategies, such as tutorial CAI the authoring of text to a screen becomes a major task for the author. In this instance, the screen must be viewed simultaneously as the authoring surface and the delivery surface.

The authoring of text does require many features which are available in word processors, however, the substitution of some word processing functions can encourage authors to approach the authoring of text in the same manner as authoring a textbook. The result is justifiably open to the criticism that courseware is often just 'page flipping'. The capabilities and user strategies of using a textbook are markedly different than the capabilities available with a volatile screen.

6.2 Text Parameters

Text authoring parameters involve at least three aspects:

a) screen characteristics,
b) font characteristics, and
c) supporting attributes.

Screen characteristics involve such parameters as the pixel resolution in both the vertical and horizontal directions and the number of usable rows and columns into which text can be placed. Font characteristics relate to the nature of the fonts available, such as for example a Greek or foreign language font set, as well as font size and style. Support attributes relate to such screen capabilities as underlining, the use of color, emphasis techniques and directing techniques such as pointers to text and enclosing text.

6.2.1 Screen Characteristics

The screen resolution in pixels, aspect ratio, and color mapping can all have an effect upon the adequacy of text displayed on the screen. A resolution which is too coarse can make the use of small characters such as superscripts and subscripts too small for easy identification. The problem of small size and readability is evident when characters initially designed for normal presentation horizontally across the screen, are displayed at an angle. The latter problem is complicated further by the aspect ratio of the screen. Since many terminals have a different number of pixels per character unit horizontally than vertically, characters may lose a distinctive font style. Encoding color characteristics of characters requires additional memory capacity in a terminal. If sufficient memory is available for encoding color characteristics for each pixel, then the problem of color "bleeding" among characters is not evident. However, where color is encoded by character unit, bleeding will occur when a character of one color is positioned across a character boundary when different colors are used for adjacent characters.

6.3 Font Type And Style

Unfortunately, manufacturers have followed what appears to be old typsetting practices of defining character font sets. Although the definition of font styles in which all characters are homogeneous in the style is reasonable, the collection of special characters as unique font sets can decrease efficiency for the author. For example, the definition of a "symbol" set containing Greek characters may be a major advantage for an author creating lessons in Greek, but a marked disadvantage to the author creating courseware of a mathematical nature. The nuisance of having to invoke a complete font change each time the Greek sigma character is required is a major disadvantage to the author when mathematical courseware is created. Ideally, the author should be able to append to the most frequently used font set those special symbols frequently used within the subject matter discipline. In addition, the authoring interface should be able to provide the support to the author by reducing the memory load of recalling which keyboard key creates the special character.

Many microcomputers with word processors provide a large variety of different font styles. Although such a variety may be useful in "desk top publishing" their usefulness for courseware development is considerably less. Special features, such as shadowing or bordering characters with other colors may add showmanship, but it is yet to be shown that such features enhance learning.

The use of special codes, e.g., text formatting codes is not recommended for authoring purposes as it requires the author to learn additional sets of commands and does not provide the author with an immediate representation of the symbols and their positional relationships. Where complex equations or character positioning is frequently required it is most desirable to allow the author to "save" or store such a configuration for repeated use.

6.4 Character Positioning

Character positioning is frequently preset as a terminal characteristic, e.g., the number of rows and columns into which characters can be placed. For many courseware applications this presents no problems, however, when the author wishes to have a pleasing relationship among elements of a complex equation containing subscripts, superscripts, symbols in the numerator and denominator, the usual row by column constraint on positioning is a disadvantage. One solution is, depending upon terminal characteristics, to provide pixel positioning of characters as a special author-invoked function. Another solution is to create the formula or equation as a graphic.

Backspacing and overstriking can be a useful feature for the creation of special characters, such as, for example the "not equal" symbol, and those fonts required in APL, although the latter may be better accommodated as special font sets.

A common problem for authors working in the mathematical area is the creation of adequate subscripts and superscripts. This process should be an easily invoked step and not require numerous key actions. Subscripts and superscripts may be created by positioning the character one half line below, or above, the current line position. However, this procedure usually provides subscript and superscript characters of the same size as regular characters being used. This implies that an additional half line spacing between lines must be created prior to, or after typing the subscript or superscript. An alternative to this method is to maintain subscripts and superscripts within the same line boundaries as the current line, but to have their sizes reduced. This however, can be a problem when the pixel density is insufficient to provide a clear representation of the subscript or superscript. Clearly, the author should not have to use program-like options such as A**2 to represent A-squared.

6.5 The Use Of Color

Color can be useful for drawing an item of information to the attention of the author, as well as for emphasis within the courseware (Sandal, 1987). However, because of the monochromatic nature of colors produced by color monitors, colors such as blue and red may be considerably reduced in intensity and difficult to recognize particularly when used as text attributes.

Changes in color should be easily made by single keystrokes when possible. However, this may require monitoring each character entered by the author resulting in delays in the appearance of the characters typed. (This phenomenon is more characteristic of terminal to host systems than in micro computers).

The use of color adds further complexity since in some processes the authoring system will be required to keep track of the current color being used. Unless a color printer is available, hardcopy documentation will not give a true representation of a screen display.

6.6 Character Size

Some manufacturers have over emphasized the capability of terminal equipment to provide a range of character sizes, e.g., 0 to 16 different sizes specifications. Frequently such a specification reduces to 1 or 2 practical sizes, all others being too small and too large. It is desirable to have size control over special symbols such as for example, the summation symbol in order to provide a clear distinction of the operator's domain over the variables to be summed. The same argument applies to other symbols such as the integral.

6.7 Dynamic Text Formatting

It is not always possible for the author to format the screen directly and there are cases where under author mode the text presented to the author will be of variable length. For example, a list of items from a library of drawings which continues to grow as the course is being authored, may be required. The exact length of such lists of information cannot be predicted. Similar situations exist during the execution of a course. In such cases dynamic formatting is required.

Ideally such formatting should be automatically done by the authoring system within the boundaries of a defined window. Where the amount of text exceeds the capability of the window to completely present the text, scrolling will be required under author control, i.e., the author may control the rate of scrolling and its direction.

Another example of the need for dynamic text formatting exists when a number has to be inserted into the stream of text. The number of digits in the number presented may be a variable and unknown at authoring time, and thus will affect the positioning of text which follows. Again, it should be possible for the author to establish a window for such displays. Without the boundaries established by a window, other items of the screen display may be damaged by attempting to display lengthy lists. Such damage may occur infrequently and cause difficulty in identifying the source of the problem.

6.8 Text Rotation

Text rotation allows the author to display text at some defined angular rotation from the normal horizontal position. Our experience suggests that this feature is not an important one, although it is useful. The rotation of text frequently presents a problem because the pixel resolution required to display characters clearly at some angles is greater than the number of pixels required for the horizontal display. An additional problem of distortion is introduced because of the aspect ratio of the rectangle within which characters are displayed, e.g., the number of pixel positions of the rectangle is not equal in both directions. Unfortunately, these problems are related to the terminal characteristics and are not easily overcome by software.

Writing text at an angle to the horizontal is probably not as useful as being able to rotate each character through 90 degrees and then displaying a sequence of text in a vertical columnar format, or to display a horizontal sequence of characters as a vertical sequence without each character being rotated. The latter feature is very useful to the author for labeling the Y-axis on a diagram.

6.9 Display Of Stored Information

There are situations which may require the display of stored information. Such information may have been acquired during the process of course execution, or from the run-time support system, e.g., the student's name. The length of the string to be displayed may be variable, and if so, will cause a formatting problem unless a fixed length equal to the maximum length of the string to display is defined. Consideration must also be given to the display of integer and real values, which should automatically be converted to appropriate form for output to the screen. If stored information is to be displayed, the information is likely to have a variable name associated with it. During the authoring process the contents of this variable may or may not have been defined.

When a fixed length string is to be displayed, this can be defined by the author through drawing a box encompassing the area. The author will have to be queried as to the name of the variable which is to be displayed. A list of these variables must be kept and checked to determine whether the variable which the author is asking to display in fact has been created.

If the variable is undefined at the time the author calls for its display while in the process of authoring, then the name of the variable may be displayed in the box. However, the variable name may have to be truncated in order to fit within the area defined. In this instance author action to "explain" or provide the complete name is desirable.

A very useful method of frequently displaying complex equations is to define the string required and store it as a named string which is then called for display by the author.

Variable length information which is to be displayed presents a problem. The run-time system is likely to have a limit to the length of the string which will be displayed, and if lengthy, a display in a scrolling window provides the best solution. Therefore, the author must have some manner of indicating that a window with scrolling features is required. If a window is not available, but the string length is finite, then the length of the string displayed and its starting position should be available to the system so that the starting position for display of subsequent text can be calculated.

6.9.1 Screen Arithmetic

The saving of parameters such as the starting position and string length of information displayed as noted above, implies that these parameters will be used arithmetically. As far as possible, arithmetic calculations to determine the starting position of subsequent text should be handled automatically. However, there are situations where the author may wish to calculate where subsequent text is to begin. If this is the case, then the author must be given opportunity to define the arithmetic function required to make the calculations. This also implies that the author may require the facility to state that "this text is to be placed according to the position calculated arithmetically".

6.10 Other Features

Because some hardware features of the terminal lend themselves to easily create blinking characters and inverse characters, i.e., characters which are written against a colored background, these characteristics are frequently incorporated in some software interfaces. Although there are situations under authoring mode and during course presentation mode in which these features can be useful, they are not considered to be features of a high priority. Blinking and inverse characters may be useful in some situations for bringing the displayed text to the attention of the author or student. A good example of its use is the blinking cursor which helps the terminal user quickly locate the position of the cursor when a screen display contains a large of number of characters.

CHAPTER 7

AUTHORING THE SCREEN: GRAPHICS

7.1 Introduction

Creating graphics to support instruction is likely one of the most complex features of any authoring system. The type of graphics required is very much dependent upon the nature of the subject matter, the author's style of presentation, and the characteristics of the student. Graphic requirements may range from simple line drawings as might be required in a lesson for an elementary school student, to the complexity of a meteorlogical map. In addition, there is a need, over a wide range of subject matter and student levels, to be able to plot data points. The manner by which such drawings are created may require more than one mode of operation. For example, in some instances it is useful for the author to create drawings directly on the screen and in the context of existing text being displayed. In other instances it is more useful for the author to create drawings independently of an existing screen containing text.

The features required, or which may be useful to the author, can be very extensive, e.g., shading color and density, line widths, curves, arcs, boxes. In some instances, where realistic displays are required, a photographic-like quality is demanded by some authors, for example, the subtle color changes in an electropheresis experiment. In some subject matter areas, such as electronics, the definition of a set of symbols representing electronics components, and connecting these together to form a circuit diagram is important. This latter example suggests that graphic entities which may be quite simple are desirable within the context of a menu.

Another feature which is very useful is the capability to create graphics and to store them in a library of graphics for later reuse by the author or by other authors. This implies, however, that a system of identification, documentation and retrieval is required, and that the graphics are relocateable to any position on the screen. (The latter implies the use of relative addressing in the creation of the graphic).

Our experience and research suggests that it is unwise to assume that a single graphics "package" can satisfy all course authors. It is far more likely, that different graphic facilities will be required by authors working in different subject matter areas. (Basic graphic elementary operations may be identical, but structured in different ways for different authors and content areas).

It is important to point out that instructional graphics are different than "advertising" graphics. The former are designed to enhance learning, are generally "noise" free, and simple. The latter, frequently provided by microcomputer graphics packages, may be "flashy" with considerable "pizzaz" but add little to enhance instruction. Wainer and Thissen, (Wainer & Thissen,1986) have discussed such characteristics of graphics in the context of plotting data.

7.2 The Hardware Interface Problem

Our experience suggests that authors become accustomed to the creation of graphics in the context of their own subject matter and extrapolate this experience to all other subject matter areas. The author who creates graphics using a mouse, seldom appreciates the problem of using the mouse for creation of an exact representation of a mathematical function. The author creating representations of mathematical functions seldom appreciates the fact that his software tools will not help in the creation of an electronic circuit diagram. Clearly, a variety of software tools are required.

The mouse or stylus of a bit pad are most useful for free-hand drawings. They are not the tools which are desirable for the representation of mathematical functions because of the irregularities which are introduced during the drawing process, or for the plotting of data points as might be used in statistics. Large cross-hairs are useful when alignment of one coordinate is to be made relative to another coordinate. However, cross-hairs are of very little use in the creation of a free-hand drawings.

7.3 Graphics In Support Of Text

An author will find it very useful to be able to combine within the context of one screen both text and simple graphics such as lines, arrows, boxes, and circles. For example, attention of the student may be drawn to a particular component of an equation by pointing to the component with an arrow, or by creating a colored box around the component. This facility is required as a natural extension of the system which allows the definition of the text to the screen. Defining a box by the parameters associated with text, such as for example a textual entity defined as a paragraph, is not particularly useful and sufficiently flexible for author use. The author should be able to simply leave text definition mode, and through the use of cross-hairs or a bit pad stylus, define the upper left corner and lower right corner of the box to be created. It is important to note that errors will be made in such definitions, and provision must be made to "erase" and redraw such graphics.

7.4 Simple Graphics

The creation of simple graphics which include lines, rectangles, arcs, circles and ellipses are useful in a wide range of subject matter. The capability to position an ellipse at an angle to the horizontal axis of the screen is also useful, although many systems do not provide for the rotation of the major axis of an ellipse.

Attributes for drawing simple graphics will vary with the terminal capability. Usually the drawing attributes include color, line type (solid, dashed, thick, thin), and shading of an enclosed area. Some microcomputers have excellent capability for shading enclosed areas, providing different shading patterns as well providing for a simple identification of the area to be shaded.

7.5 Graphics And Special Fonts

The capability of providing for the use of special character fonts as part of a drawing is very desirable. For example, an author may need to label an item of a drawing in another language, to label a graph using Greek symbols, or to use subscripts and superscripts as required in some area of mathematics. The author should not have to "stand on his head" to invoke font changes or to make use of subscripts and superscripts when these are required in the context of a drawing. It is also useful to be able to use different sized fonts, for both regular font sets and special font sets.

7.6 Plotting Data

Numerous software systems are available for plotting of data. Some systems are designed to produce an accurate and precise representation of data as would be required for inclusion in a scientific technical report. Other systems, particularly on microcomputers, seemed to be designed for the creation of 'eye catching' displays to be used at board meetings! With regard to the latter type of plotting software, Wainer and Thissen (Wainer & Thissen, 1986) have the following suggestions:

> "Don't add anything else. No extra dimensions, no extra colors, no ancillary images, no pomp, no circumstance, no noise, no schmutz, nor schmerz, no accident, no catharsis, no chance, no humbugging, no button-holing, no exploitation, and no mixing things up."

The above comments are even more applicable to the design of instructional graphics where clarity of principle is of primary concern. With this in mind 4 basic steps can be identified for the production of instructional plots:

1. defining the form of the x-y axes and their attributes,
2. defining the plot characteristics,
3. defining the plot type,
4. identifying the source of the data, and
5. labeling the plot.

The points noted above must be considered in relation to the degree of author interactiveness which is desired. The degree of interactiveness will vary. For example, an author may easily create a simple plot "on the fly" if the nature of the representation is not complex, e.g., a simple bar graph. However, where a mathematical function is being plotted it is not likely that an author will wish to enter all the data points from the terminal, but rather call upon a file of data points. This also implies the existence of data files which can be used repeatedly for the plotting of data. Files which are stored, must be documented, and retrieved for repeated use by the author. For any non-trivial courseware, hundreds of plots may have to be stored. As an alternative to the storage of data points, the mathematical function which would create the data points might be used. This has the advantage of providing for a wide range of functions, but would require an appropriate function interpreter, and for some complex functions, would force the author to wait for the data points to be created. A combination of being able to create the data points through the definition of the mathematical function, and storage of the data points for later use may provide the optimum environment.

7.6.1 Defining The Axes

The author need only be constrained by the condition that the axes created must intersect. With such a general constraint, a wide range of axes may be defined, not just those enclosing the upper right quadrant. These axes should be plotted interactively when being created in the context of a screen of text so that the author may ensure that the final plot including any titles fit the available space. In addition, having the ability to interactively define the length of each axis allows for replotting of data into an area defined by the author. The characteristics of the axes should include color, and the color black should be allowed in order to create plots in which the x and y axis are not visible to the student.

The number of units into which each axes is to be divided should be a variable defined by the author. The identification of the numerical value of each division should not be automatically displayed, as frequently these may take on such obtuse fractional values as to be confusing to the author and student. Frequently, the author may wish to simply label units with characters and not numbers.

If the author wishes to stop after the creation of an axes without actually plotting, this should be possible. This allows for the display of an axis required when the student is asked to identify coordinate points during courseware execution.

7.6.2 Plotting Characteristics

The major plotting characteristics are the available colors and the value of each marked unit of each axis in terms of the metric of the data being plotted. Minor plotting characteristics might be defined as line type (solid of varying thickness, dashed, etc.) Characteristics such as shading should not be defined as a general plotting characteristic as frequently the shading is desired for only a portion of the plot, as for example when the area under a curve is being illustrated. Such a feature is enabled and disabled as a function of a specific data point.

7.7 3-Dimensional Plotting And Rotation

For some applications 3-dimensional plotting and rotation become important, although for some complex functions the generation of data points, particularly where contouring features are desired, can be time consuming and require considerable computational power.

The capability of at least simple 3-dimensional point plotting with rotation to allow an author to select the view he wishes a student to see in the context of courseware is desirable. The division of the plotting functions should be as noted earlier, i.e., defining the axes, the plot parameters, etc.

7.8 Animation

The ability to provide an adequate animation depends very much upon the characteristics of the terminal. Some animations would require considerable computing power and speed. Such capabilities are becoming available in some micro computers, but are generally not available for terminals dependent upon computational power of a host system unless high data transmission rates are available. Authors frequently underestimate the amount of computing which is required to produce an animation which is compressed in time and viewed through the medium of film.

In addition to all the features required for plotting, animation requires the definition of additional parameters. Some of these parameters include:

1. the definition of a travel path of the object
2. the definition of speed and acceleration of the object
3. the definition of rotation in 1, 2, or 3 dimensions

CHAPTER 8

TEXT AND GRAPHICS EDITORS

8.1 Introduction

Editors are required by the author in order to make changes to textual and graphical displays. Although guided and controlled authoring through software control can minimize the number of errors made by the author, software cannot control the need for changes which are sought in order to optimize the effectiveness of courseware. Optimization of courseware is required after pilot testing. Although there are advantages in considering an authoring system as one large editing system, and this approach certainly has merit, a considerable amount of complexity is introduced since functions of a text editor as well as a graphics editor are required. Frequently the simplest option is to make use of editors which are provided by the manufacturer, even though these may be distinctly different for text and graphics editing. There is considerable merit in being able to access the internal representation of the course through more than one editor. This point is also important to the designer of the authoring system as it provides a "second door" through which problems can be isolated. The latter may imply examination of "source code", but there is merit in this capability (Weiser, 1987)

8.2 Compromises And Issues

The design of the text and graphics editors requires that some compromises be made. The following factors are important in this regard:

1. the programming language available in which the editors are developed
2. the nature of the terminal configuration used by the author, and its hardware and software characteristics
3. the amount of local and remote (terminal to host) processing which is required
4. the development tools and resources available

Invariably the authoring system must interface to a terminal with defined characteristics even though the authoring system maintains an internal representation of that which appears on the screen independent of a specific type of terminal. Maintaining an independent representation of such information provides the possibility of terminal independence for the author and the student during courseware execution, however, there is considerable increase in design complexity. Of particular importance is the design and manipulation of data structures which hold the internal representation since it is the contents of these data structures which will have to be modified during the process of editing. In the case of graphics, not only does the representation which creates the object have to be maintained, but it must also be possible to erase components of this representation. In addition, a transformation of the data within these structures has to be made to accommodate a specific type of terminal. An alternative to this position requires the storage of the code used directly by a specific terminal to eliminate the transformation step required to accommodate a specific terminal.

An alternative which may have to be considered initially is to use general line editors provided by manufacturer and to edit the code directly if the authoring system generates code in human readable form. This has the marked disadvantage that the author must understand the code which is being edited as no screen representation is available during the editing process.

Ideally, generalized data structures with the necessary software controlling the use and modification of the contents of these structures is the preferred solution, although more expensive to create initially. In the long run such data structures can be used for the storage and modification of a wide variety of information, not just textual and graphical information.

8.3 The Edit Space

Some authoring systems use a frame oriented system in which a complete screen is defined as an entity and maintained internally as such. The same systems allow more than one frame to be displayed at one time by defining windows, which in effect provides for smaller frames to be displayed at the same time. This considerably simplifies the definition of the edit space, i.e., the domain of the internal representation which must be considered for modification during the editing process. The definition of the edit space becomes problematic when free formatting is allowed the author, and a screen display at a particular point in time may represent the accumulation of several write operations to the screen. Some components of the screen may have originally been written many steps earlier and from many different locations in the course. If these components are to be edited, then the editor must be able to identify the exact internal representation which will require modification. Possible solutions to the problem include the following:

1. tag units displayed on the screen so that their internal location is known

2. provide for the use of a large number of windows

3. provide for editing only when the representation is delivered to the screen, since at that instance the source of the internal representation is known.

8.4 Text Editing

Text editing is required during the process of formatting the screen and during testing phases of courseware development. The edit space is likely to be clearly defined during the time the screen is being written to and at this time editing presents fewer problems than editing during an execution sequence.

The primary text editing sequence involve changing the following:

1. one or more characters which may also involve a font change, e.g., the use of a Greek symbol

2. position of the text

3. color

4. size of the text

5. slope changes

6. special control characters if they are used, e.g., control characters which allow the creation of special equations

7. superscripts and subscripts which may involve the use of a special font, and/or involve the use of index and reverse index operations, as well as the backspace.

8. characters formed by overstriking one of more characters.

The points noted above imply that there is not a one to one correspondence between what appears on the screen and what is contained within the internal representation of that written to the screen.

Generally, editing during the time the text is being created is not a problem as most hardware provides for the use of a delete or backspace key. However, if special function keys are used such as the escape key, these keys may prohibit deleting beyond the point at which they were used. The only solution is to monitor each keystroke so that special function keys can be identified. On some systems, this could reduce the speed at which software can react to the input.

8.4.1 Text Positioning

Ideally a visual editor is desirable for the editing of text on the screen. Positioning the cursor either by special function keys, or a mouse works well. However it is more complex to move blocks of text on the screen since the boundaries of the block must be unambiguous. For example, some software provides for "dragging" a cursor across the text to be moved with the boundary being identified by the text appearing in reverse video format. The capability of changing the position of the text in the manner described depends very much upon how the text is represented internally, i.e., the data structures holding the text, and whether the text is stored in memory or moved to some secondary storage device. (We must keep in mind that not all text appearing on the screen was written to the screen in a sequential, top to bottom manner).

Text re-positioning may be considered part of a more general feature of "cut and paste" which involves identification of the domain over which the action is to take place, a removal, and a repositioning.

8.4.2 Color Changes

Color changes during text entry to the screen should be available with simple keystrokes. For terminals which are capable of providing for a wide range of colors and intensities, more than a single keystroke may be required, e.g., parameters may have to be entered with a color shift. When many parameters may be required, then a menu of options is useful although identifying all the parameters may be time consuming for the author. In this case default values, and/or author specified combinations which become part of a menu and which can be associated with a special function key are recommended.

8.4.3 Size Changes

The effectiveness of the various size of text provided by the terminal hardware has been noted earlier as being far less useful than manufacturers suggest. Changes in size should be accommodated through special function keys or through a menu. The variation in font style is not considered an important variable from an instructional point of view. That is, clear and easily read font is satisfactory. One does not need to accommodate shadowing effects, degree of boldness, etc.

8.4.4 Slope Changes and Rotation

Changing the slope of a string of characters is most useful for such purposes as labeling the vertical axis in a graph. Creating and modifying such strings is not readily available even in reasonably well designed word processors because of input hardware is designed for left to right horizontal deposition of characters. Generally, vertical text can only be accomplished by considering each character as occupying a separate line. Slope changes should be accommodated through a default attribute of the text menu.

8.4.5 Editing Special Control Characters

The editing of special control characters can be a problem because they may not be visible to the author, e.g., a backspace is generally not a visible character. If provision is made to make such special characters visible, then the screen layout is disrupted unless it is possible to "cut out" the text having such characters, place it into another part of the screen where the non-displayed characters may be viewed and modified, and then returned to the appropriate position on the screen in order for the author to assess whether the screen format is still acceptable.

8.4.6 Editing Overstruck Characters

Overstruck characters may be required where symbols such as those used in APL are required. In this instances it is not always possible to determine which of two characters was struck first. Positioning the cursor on the character and the use of a delete operation is the only manner by which the order of the characters can be determined, otherwise a separate representation has to be displayed.

8.5 Graphics Editing

Editing graphic screen displays is by far the most complex of editing procedures. It is complex not solely because of the problems which exist in text editing, but because of the many attributes and components which together form a graphic. A graphic may be composed from a composite of such components as lines, arcs, ellipses, and forms derived from a mathematical definition. In addition to such common attributes as color, there may be line thickness and type (dashed, dotted, etc), as well as shading of closed areas. Editing of these components and attributes requires the isolation of a particular component and its attributes from other components and their attributes. Clearly, the flexibility of editing each component depends very much upon its internal representation.

Components and attributes may be codified for an internal representation, or it may be represented as a bit pattern of the screen. If the representation is a bit pattern then the edit space is unambiguous, i.e., it is that which can be observed on the screen. However, if an internal graphics code is used, then the edit space is not as obvious. For example, a circle, and there may be more than one on the screen, has to be isolated from other components before an editing action can be taken.

If a bit map representation cannot be maintained, such as is likely the case in a terminal to host configuration, then a controlled step-by-step re-creation of the screen can allow the edit space to be precisely defined and thus changes made without encountering ambiguities.

The creation of some graphics can be of such complexity as to warrant the use of a separate graphics software packages which can be invoked from the authoring system.

8.6 Editing Plotted Graphics

Simple plotted graphics (data plotting) require the same editing capabilities as for editing graphics. However, for some changes replotting the graph may be easier for the author to handle than to attempt making specific changes to such components as the axes and the function itself. In the case of editing titles and labels, those procedures used for text editing would be most useful.

8.7 Multiple Editors Vs A Single Editor

Our work suggests that a single editor can be constructed which would handle both text and graphics. However, the software is complex and requires very careful definition, and access to the definition of the data structures holding the internal representation. Given a set of basic editing software functions and information of the data structure containing the internal representation of the object to be edited, a single editor can be created to handle text and graphics in a controlled manner.

CHAPTER 9

AUTHORING THE STUDENT RESPONSE MODE

9.1 Introduction

Authoring the student response mode is the process by which the author conveys to the system that a student response is required. Usually, student input is taken as a typed response using a keyboard, or a pointing response. In the case of the pointing response the student may have the facility to use a set of cross-hairs, a mouse, a light pen, or to simply make a touch to a touch sensitive screen depending upon the terminal characteristics. On some occasions other input devices may be used which provide input to the system such as, for example, a measuring instrument which the student is required to adjust.

9.2 The Nature Of The Response

The student's response may be a string of characters or digits, or a combination of both as might be the result of a pointing response. If the response is made as digits consideration may have to be given as to whether these digits are to be treated as characters, a real number value, or an integer value. These considerations may be taken care of during the response analysis phase, and be taken in as a string of characters for later conversion to an appropriate form.

The student's input may be required in a special font such as would be used in a foreign language or for a mathematical parameter. In this case, the student must know which character on the keyboard is to be used. The same problem is raised when the student is expected to input a mathematical equation which may require the use of superscripts and subscripts. If small-sized characters are not available for superscripts and subscripts, then consideration must be given to the use of the index, reverse index, and backspace key operations. These may not necessarily be used in a consistent order by all students and result in the need for special response analysis algorithms.

For some subject matter it is important to allow the student to input a response in which the characters require a unique spatial orientation, such as for example, in organic chemistry. In the latter case not only is the spatial orientation important but also the use of subscripts may be important. A spatial orientation is likely to require the use of more than one line of input by the student.

In addition to the use of the keyboard for input, the author may wish to have the student make a pointing response to identify one or more words on the screen or some component of a graphic, or for identifying a X,Y coordinate. A pointing response can be important for use with young children who cannot construct a keyboard input.

The author may wish to have the student input a simple drawing of some type. This becomes a complicated problem at the response analysis level. However, there are instances, such as in geometry where the author may wish to have the student make a simple drawing, e.g., a right angle triangle, two parallel lines, etc. In these instances the author must be able to convey to the authoring system that the student input is to be a drawing.

Even with the input of characters, authors frequently wish to define the format of the input to be of free form, of a multiple choice or true/false type, or fill-in-the-blanks. These formats are frequently used during examinations.

The specification of the parameters required to define the nature of the input when the various forms of input as noted above are considered, adds a measure of complexity to the authoring system. Clearly, an authoring system which allows only limited input such as single digits or characters for a multiple choice response, is inadequate for most non-commercial instructional applications.

9.3 Identification Of Student Input

It is important that some of the parameters associated with the input are defined by default. These parameters are to be associated with the input made by the student and are later used to associate the response with the question posed, or more generally, the exact location in the course where the input was encountered. A unique identifier must be associated with the input. An additional identifier might be used to indicate whether a record of the input made should be saved. Our recommendation is that the identifier should be defaulted to an identifier derived by the system to ensure uniqueness. If author defined identifiers are used, then a list of such identifiers must be kept in order to prevent duplication by the author. Although for courseware of trivial length the author may use meaningful identifiers, e.g., problem 1, start, end, repeat, diag1, etc., in a course of any useful length the author cannot maintain the uniqueness without system support. The system support requires storing these identifiers for presentation to the author, and prevention of duplication.

Consideration also has to be given as to whether the student's input should be saved or not. For example, it is likely during the optimization phase of courseware development and during examinations, the author will want a record of the student's input. Writing out the response may be considered part of the definition of the student input, but is perhaps better considered a separate authoring action following the definition of student input.

9.3.1 Definition Of Keyboard Input

The following attributes and characteristics need to be considered as capabilities for an authoring system in order to define the process of student input:

1. the position on the screen where input will be made
2. the length and width of the input area
3. termination of input e.g., carriage return, number of digits, or a time parameter
4. the font set to be used, e.g., special characters
5. color of the input text

In the simplest case when single characters or digits are to be input as a response by the student, the author can define the position by moving the cursor to the position. However, this restricted mode of input is generally inadequate. The author must be able to convey a number of parameters to the system. These parameters include the beginning point of the input, the length of the input, the width of the input since the input may involve superscripts and subscripts, the font set, the color, and the nature of how the input is to be terminated. A menu with default conditions is recommended. Default parameters would be color, and width of the input (one line), carriage return as the terminator, and/or time, and/or the number of characters or digits input whichever is encountered first. (It is desirable of course, that the run-time interpreter be able to distinguish the nature of the terminator which was used by the student).

Although the specification of the length and width of the input may appear restrictive they are necessary in some important instances. For example, if a student is asked to complete a table of values arranged in columnar format, it is important that the number of digits input be restricted to the width of the column so that the columnar format is not damaged.

The author requires some confirmation that the area defined for the student input is adequate relative to the space available for input on the screen. This may be accomplished by defining a rectangle encompassing the area. Since the defined area may be found to be inadequate by the author, the author should be able to immediately correct the definition. From such a definition, the maximum length and width can be calculated if required, and the area is more likely to be adequately defined. Default parameters would still be maintained for defining the termination of input, color, and font set. When a special font set is used, consideration must be given to letting the student know which keys of the normal keyboard have been associated with the special font set.

9.4 Defining A Pointing Input

Authors frequently require a pointing response be made by the student. Very young children who have not yet learned to read and construct words or phrase may for example, be requested by an audio message to point to a graphic object, or in trade training a student may be requested to point to a specific component of an electrical circuit.

The terminal hardware may be an important consideration in what attributes are to be associated with the input. For example, if a light pen is used, since its activation depends upon receipt of light from the screen, an unlit area of the screen cannot become a target for the pointing response. A graphic object may have many unlit areas. If the terminal does not have a specific device to provide for a pointing response, the cursor may be positioned to the target area, although the resolution then becomes equivalent to the area of one character.

If a pointing mechanism such as cross hairs, or cursor controlled by a mouse is available, consideration also must be given to the resolution of the movement and speed with which the cursor can be moved. The movement of cross hairs in pixel units could be very slow, while the use of a mouse is generally quite fast.

Regardless of the nature of the pointing response, an author will need to define a target area which encompasses a domain slightly larger than the target area. Thus provision to define the boundaries about the target area become important.

The most efficient manner by which the target boundaries may be defined is for the author to construct the rectangular space within which the target exists. In some instances a circular area may be required. The boundaries of the target area should be continuously visible to the author so that the boundaries of any subsequently defined areas may be appropriately defined.

9.5 Graphical Input

It is conceivable that an author may wish the student to draw an object in response to a question. Although the definition of the input may be straight forward, the analysis of the object drawn is considerably more complex since the nature of the object drawn may be unlimited. One approach to this mode of input has been considered by Lee (Lee, 1983) which requires that the student be restricted to the creation of only certain basic components of an object, e.g., straight lines, arcs, circles, etc. Such options may be selected by the author from a menu list. Providing the student with a known (and perhaps restricted) set of drawing moves simplifies considerably judging the adequacy of the response. For example, a triangle cannot be drawn without using a straight line drawing option. Complexity still exists in determining the adequacy of the object drawn, e.g., a right angle triangle may be oriented in an infinite number of ways on the screen, the three sides may not represent a closed area, the right angle may not be exactly 90 degrees, the size of the drawing may vary, etc.

As with the input of a string of characters, some provision must be made for the student to define the termination of the input. Usually a target word such as "quit" is adequate.

CHAPTER 10

AUTHORING FOR ANSWER JUDGING

10.1 Introduction

Judging the adequacy of a response from the student will continue to present a major challenge for CAI as well as for the design of CAI authoring systems. The problems related to answer judging raises the critical question as to whether an authoring system can have general procedures applicable to a wide range of subject matter and methods of response input. The nature of the procedures which might be required by an author creating courseware for use by grade 3 students in the area of arithmetic are not likely to be very useful to the author creating software for use in second year biochemistry, unless both use trivial procedures for the judging processes.

10.2 The Nature Of The Input

Consideration must be given to the nature of the input by which a student will provide a response. The attributes of a response may include:

1. input of a character or string of characters
2. input of digits in integer or real format including vectors and small matrices
3. a pointing response
4. a graphical response
5. special characters e.g., superscripts
6. a combination of the above with positional or spatial orientation.

10.3 String Input

Depending upon the nature of the terminal equipment and software available, it is possible to treat character and digit input as simply a string of characters. Indeed, special characters such as cursor movements, if required to create subscripts and superscripts, may also be considered as part of the input string and subject to consideration in the response analysis algorithms. Under some circumstances, the coordinates returned due to a pointing response might also be considered as simply a string of characters. However, it is unlikely that graphical input could be usefully considered as a string of characters representative of the object which was drawn.

10.4 Numerical Input

The input of digits which are represented directly as real or integer values allows for the evaluation of the response to be made in numerical terms. However, there may also be positional characteristics such as when superscripts, vectors, and matrices become part of the input. The evaluation now becomes more complex since it involves more than a numerical evaluation. Numerical input may also include constants which are within the context of the subject matter but represented as characters. e.g., some constants in physics, the Greek symbol π, the mathematical constant represented by the character e, and in subject matter which involves trigonometric functions, sine, cosine, arcsin, etc. Clearly, "numerical" input is more than the input of digits which can take on a numerical representation. Thus, for numerical input the authoring system will likely require the capability of obtaining from the author specification of real and integer values as well as special words which have a mathematical meaning. For purposes of specifying the analysis of responses, the authoring system will have to be able obtain sufficient information from the author to assess real and integer values within some degree of tolerance, and also to identify and assess the adequacy of special character strings.

10.5 Pointing Input

Generally a pointing response returns a coordinate indicative of that point of the screen which was identified by the student. These may be returned by the hardware as numerical integers, real values, or a string which contains digits, delimiters (such as commas) and closing brackets or parentheses. In this instance an author must be able to also define the tolerances which are to be used in judging the adequacy of the response.

10.6 Graphical Input

By far the most complicated input is that of a graphical input since the variety of the objects created is essentially infinite. The nature of the internal representation of the input becomes very important since it may very much constrain how the author may analyze the input. Graphical responses may be stored as a bit map by the system or have some internal character representation. The attributes which may be present even in simple objects may be difficult to evaluate. For example, how close must a sine function drawn by a student be collinear with a correct representation of the function? Graphical input also requires the author to define that the mode of input is to be graphical. This implies that the terminal must be activated for a drawing mode.

10.7 Typical Solutions

Because of the complexity of the problems associated with analysis of student responses and the associated constraints of CAI systems, authors tend to avoid asking questions which would require complex analyses relative to the processes available. For example, rather than asking a student to draw a sine function, the author is likely to present several plots of trigonometric functions and ask the student to select the appropriate plot.

Perhaps of more importance, we seriously doubt that one set of generalized procedures can be defined to easily and efficiently accommodate the processing required by most authors in all subject matter areas. Some general procedures can be defined for preparing the student's response for evaluation, but the complexities of actually evaluating the response, including the use of parsing procedures suggests that unique procedures will have to be provided for different authors. Although some AI and expert system techniques hold promise for solving the problem, they currently are not practical in the real instructional environment and non-trivial course delivery environments, and certainly are not easily defined by authors having as their goal the production of usable courseware within some reasonable period of time.

10.8 Processing The String Input

Specifying the procedures for evaluating a string input can become a complex processes as the procedures are dependent upon the nature of the input as well as the characteristics of the subject matter. It is reasonable to expect that the evaluation of an elementary student's response to a spelling test will be different in nature and complexity than the evaluation of a definition in a linear algebra course. However, some common procedures are recommended and are noted below.

It is useful to considers the evaluation of string input as falling into the following five steps:

1. defining the comparison standard
2. cleaning up the input string
3. transforming the input string and the standard
4. evaluation of the input string against the standard; parsing may be required
5. making a decision based upon the evaluation

Consideration must be given to the processes defined within and among these classes being invoked individually as well as sequentially. This implies that the results of one step must frequently be acceptable as input to the next step. In addition to the five steps noted above, the input string may be evaluated by an author defined parsing procedure, and this procedure may also invoke a numerical evaluation as might be required when the student has input an equation.

10.8.1 Cleaning Up The Input String

The author cannot assume that input from the student is free from noise, i.e., characters which are redundant or considered to be irrelevant by the author. Cleaning up the input can considerably simplify the application of subsequent processes used to abstract meaning from the input. In some instances the trapping of special function keys, defined by the author will be required. The following procedures are recommended for selection by the author:

1. removal of leading and trailing spaces
2. removal of irrelevant words contained in one or more author developed and identified lists
3. removal of redundant spaces
4. removal of characters which negate each other, e.g., a backspace followed by a space
5. change alphabetic characters to lower case or upper case (or do not change)

These characteristics should be presented through a menu to the author and remain in effect until changed. The author should be able to break a sequence of authoring activities and have the menu displayed to confirm or to change the settings.

It is important for the run-time courseware interpreter to automatically report whether a particular cleanup procedure resulted in a change to the original input.

10.8.2 Transformations Of The Input

A more generalized approach to cleaning up the student's input is to consider the clean up procedures as part of a more general procedure of transforming the input. For example, it is possible to considered the removal of redundant spaces as a substitution of 1 space for 2 or more spaces. The same algorithm could then be used for example, to substitute 1 period where 2 or more adjacent periods have been entered by the student. Transformations are important because they may be integral to the nature of the response analysis which follows. In fact, some transformations may be considered in some systems as part of specific response analysis algorithms. Transformations considered useful, and which should be available as options to the author are as follows:

1. substitute "string 1" for "string 2"
2. place the characters of a string into a spatial orientation by assigning a new line of a matrix for each index and reverse index.
3. convert to a phonetic string
4. convert word representation of numbers to their equivalent digits
5. remove vowels
6. extract "string1" and substitute "string2"; if string1 is present it is saved as the current response input
7. case changes

It is important that the student's original input be preserved for later use by the author and the system in case transformations after the first are not to be applied cumulatively. This implies that the authoring system must allow the author to repeatedly access the original input.

Of those transformations listed above, the most useful will be the substitution of one string with another. For example this transformation can be used to remove a word and replace it with a number of spaces equal to the number of characters in the word removed, or to remove a word and compress the result, e.g., remove "the " and replace it with the null character "".

Placing the characters of the input string into a spatial orientation can be most useful in situations where the student's response is an algebraic equation containing superscripts or subscripts, and when for example, the student constructs a molecular arrangement in organic chemistry, e.g., carbon tetrachloride as a carbon atom surrounded by four chlorine atoms. In the construction of this input the student must position the cursor to place the chlorine atoms about the carbon atom. Placing these characters in a 2 dimensional matrix will later allow answer analyses to proceed on the adequacy of the spatial relationships. In the case of an equation, such as for example $y=x^2$, the spatial arrangement allows rapid identification of the presence of a subscript.

10.9 Defining The Standard For Comparison Of String Input

A response must be judged relative to an explicit or implicit standard defined by the author. An explicit standard is defined when the author lists those strings or characters which are to be sought in the input. An example of an implicit standard would be one defined by the end result of parsing the input string and tracking through the associated tree structure.

Standards defined by the author may include keywords in the simplest case, however, authors are likely to want to be able to specify the logical relationships among units, for example logical relationships among keywords of the standard. The definition of one or more standards for assessing the adequacy of a string response is required. In some instances, one standard may be used much more frequently than another standard, as for example, when a student has many options which allow access to a glossary of terms. Thus, the author should be allowed to store standards which are used repeatedly in a bank (likely a list structure of some form). Such a bank would allow the author to store sets of standards and their defined names and preserve the uniqueness of these names. Provision is also required for the author to review the items in such a bank.

10.9.1 Processing The Standards

Those steps noted earlier for processing a string input should be available for application to the definition of the author's standards. The reason can be illustrated with an example in which the author expects the two words sodium chloride as a response. The author should not have to separately define the following answer standards:

1. sodium chloride
2. Sodium Chloride
3. sdm chlrd
4. Sdm Chlrd

The difference between the first and second standard and the third and fourth standard is a case shift, while the difference between the first two and the last two is the removal of vowels. Since the case shift and removal of vowels are transformations which will be applied to the input they would also be applied to the standard defined by the author. Thus, the author would only be required to define "sodium chloride" once and not each variation.

10.10 The Transformation Report

It is desirable for the run-time system, and indirectly for the author, to know when certain transformations were applied, and to know whether the transformation effected any changes.

It is suggested that three reports be made:

1. whether the transformation made any change in the string
2. the word position in which a transformation was applied
3. the character position in which a transformation was applied

The reports may be binary vectors of varying length. The sums of these vectors can be used in defining the nature of the feedback provided to the student. The vectors at the character position level can also be used for masking operations. In order to discuss the possibilities in the use of the reports, a more formal symbolism is introduced below. However, the symbolism is not representative of what the author would have to use.

Let the general form of specifying a transformation be:

TRANSFORM(type_name,string)

The terms are defined as follows:

1. type_name is the name of the transform, e.g., remove two or more adjacent spaces
2. string is either the student's input string or author standard

The transform makes a report which may be evaluated by the EVALUATE process which may include specification of an appropriate branch. The EVALUATE process will at this point be simply defined as one or more processes for evaluating the application of a transformation or the result of a matching procedure relating a standard to a student's string response. For purposes of illustration, we shall define the following:

1. REPORT; is a binary variable (1 or 0); 1 indicates that the transform was carried out somewhere on the input string

2. REPORT_1; is a binary vector in which a 1 indicates that the transform was applied to a word and a 0 indicating it was not applied to the word

3. REPORT_2; is a binary vector made up of groups of ones and zeros; a group of ones and zeros indicate the result of the transform as applied to each character of the word; word groups are delimited by a space.

(Of course, REPORT is derivable from REPORT_2 and REPORT_1 but the author need not know how these are prepared by the system).

For purposes of illustration suppose the spatial transform is applied to the student input y=3+x**2. The student has used the Fortran programming language representation (**2) to indicate that the superscript 2 is to be applied to the x. The first transform which the author might invoke in these circumstances is to substitute the character "reverse index" for the characters **.

TRANSFORM(substitute)[(\r,**),student_answer]

The result would be a new student string containing y=3+x\r3.

The reports would be as follows:

REPORT: 1 (indicating the transform was performed)

REPORT_1:1 (indicating the transform was performed on a word unit)

REPORT_2:00000110 (indicating the 6th and 7th characters which were the ** was transformed)

At this point the author would know (because REPORT=1) that the characters ** were used, and might wish to tell the student to use a properly defined exponent in the future.

Next the application of the spatial transform would be made.

TRANSFORM(spatial)[student_answer]

The resulting reports would be as follows:

REPORT:1 (indicating the transform was applied)

REPORT_1:1 (indicating the transform was applied to the first and only word unit)

REPORT_2:000001 (indicating that the 6th character was spatially adjusted relative to the starting point of the string) If the result of the transform was stored in a character matrix such as shown in Figure 10.1.

	1	2	3	4	5	6
2						2
1	y	=	3	+	x	

Figure 10.1 Example of Spatial Transform

If one row above the starting row exists (the starting row is the row containing y=3+x), a superscript was present; the number of non-null positions would indicate the number of superscripts in the student's response.

The major problem for the authoring system is to interface the author in such a manner as to collect sufficient information to define the response analysis process regardless of what the processes might be.

Transformations should be considered as elementary operators which when properly sequenced provide a powerful series to extract meaning from the response.

10.11 The Matching Processes

A matching process is a definition of how the author's standard is to be related to the student's response e.g., elastic match, spatial match, key letter match, edit distance, etc., and nature of the report which is made (Nesbit, 1985). The author's standard may include the use of relational operators, e.g., at least AND, NOT, and OR. The reports made contain information as to the result of the process.

Matching the standard against the response may produce different information than matching the response against the standard. Consider the following examples in which STANDARD identifies the author's standard. (Since the author may have several different standards, the specific standard is identified by (ID1)). The author has been allowed to indicate that he wishes to check the student's response for the words Edmonton and Alberta, and that the response must not contain the word Calgary. The student's answer has been given as "Lethbridge, Alberta".

```
STANDARD(ID1)[(Edmonton Alberta)NOT(Calgary)]

response:Lethbridge, Alberta
```

The matching process might be defined in the form:

```
MATCH(match_type,standard_id,student_answer,match_id)
```

where match_type might be a character by character match, and standard_id the identification of the author's standard, and match_id the the identification of the result, e.g., correct.

The reports made by the matching process would be as follows:

REPORT:0 (indicating a complete match was not made)

REPORT_1: 0 1 ~0 (indicating that Edmonton was not present, Alberta was present, and NOT Calgary was absent)

REPORT_2: 00000000 1111111 ~(0000000) (indicating no character matches occurred in the first word (Edmonton), the characters in Alberta were all found, and the characters in Calgary were not found.

If the match was made of the student's response against the author's standard the REPORTs would be slightly different:

REPORT:0 (indicating a match was not made)

REPORT_1:0 1 (indicating the 2nd word (Alberta) matched

REPORT_2:0000000000 1111111 (Indicating no characters of the word Lethbridge were matched, and all characters of Alberta were matched)

In matching the response against the standard the author would learn that Edmonton and Calgary were not present in the response. We could tell the student that Edmonton should be in the response. In matching the standard against the response, the author could tell the student that Lethbridge should not be included.

If the author's reply is going to be more than identifying an acceptable, unacceptable, or unrecognizable response, the type of information embedded in the 3 types of reports are likely to be required.

Normally, REPORT would provide for the simplest author reply, i.e.,., match or no match. REPORT_1 would provide for identifying which words units need to be given special attention, while REPORT_2 would provide the basis for identifying the appropriateness of each character in each word.

It would be useful to be able to conduct some arithmetic on the binary report vectors, e.g., sum of REPORT_2, proportion of ones in each binary group reported by REPORT_3.

10.12 Diagnose

It is desirable that the author be able to identify more than an acceptable, and an unacceptable response, or to request another response if the last response was unrecognized. An author should be able to diagnose the reason for an incomplete or unacceptable answer.

The diagnose procedure is a more difficult one to define because to some extent the diagnosis depends upon what the author wishes to isolate in the student's response. It is suggested that DIAGNOSE operate on REPORT_1 and REPORT_2 and normally be contingent on REPORT. A simple example follows:

STANDARD(ID1)[(Edmonton Alberta)NOT(Calgary)]

response: Calgary Alberta

REPORT_1: (0 1)~1

From this we should be able to say that Edmonton was missing, Alberta was appropriate, and Calgary should not have been included in the response.

If the response had been Lethbridge Alberta, REPORT_1 would have been (0 1)~(0) and the author could only say that Alberta was correct, and whatever else was present was inappropriate.

If we match the standard against the response (instead of the response against the standard) the author would get ~(1)(1) from REPORT_1 for the response Calgary Alberta. The author can now say Calgary should not have been included, but Alberta is appropriate.

In these examples we need to use the binary results to select word strings from the STANDARD or from the student's response, and be able to make them available for display.

REPORT_2 can be used to show the student where character errors exist. Suppose the student's response was Edmenten Albirta, then REPORT_1 would be (0 0)~(0) while REPORT_2 would be (11101101 1110111)~(0000000). The author could use the binary string as a mask to ask the student to correct the answer: Edm-nt-n Alb-rta. An alternative would be to use the edit-distance (Nesbit, 1985) match, but this may be less appropriate for a response such as y=3+2x.

The use of REPORT_1 suggests that the following lists of word strings needs to be produced by a DIAGNOSE process and isolated for insertion into an output string:

1. words which were appropriate (words_ok)
2. words which were missing (words_miss)
3. words present but should not have been present (words_to_exclude)

DIAGNOSE_1 might operate on REPORT_1 for word selection, and DIAGNOSE_2 on REPORT_2 for character selection.

10.13 Definition Of Evaluate Process

In order to gain a further appreciation of the task an authoring system has with regard to interacting with the author for the definition of response analysis, the information in the three types of REPORTS must be evaluated, and a decision made as to the next instructional move.

The process of evaluating the REPORT (1,0) produced by the match or transform process determines whether a response has been identified. Since the action to be taken is likely (but not always) to be different for 1 and 0, specification of branching must be made.

For purposes of illustration, we shall define a form for the evaluate process, not as it might be viewed directly by the author during an authoring session, but as it might be represented internally within the authoring system.

Consider the general form of the evaluate process as follows:

EVALUATE(REPORT={1,0}/{NEXT,EXIT_ANSWER,CONTINUE,PROCESS_NAME})

where one of the components or actions listed between { and } must be defined.

1. NEXT means the next process in the sequence of processes
2. EXIT means to exit the answer analysis sequence
3. CONTINUE means to continue to the next statement
4. PROCESS_NAME means to continue to the named process
5. RETRY means to re-execute the request for a response

For example,

EVALUATE(REPORT=0/PROCESSA)

would mean that when REPORT equals 0, PROCESSA should be carried out next. This general form will be used to note the nature of the actions the authoring system must obtain from the author.

10.14 Display

Normally, an author will wish to provide some comment about the nature of the student's response. For purposes of illustration, the DISPLAY process will be defined for this purpose.

DISPLAY refers to the text to be displayed, normally following an EVALUATE step. DISPLAY may have a modifier, eg., DISPLAY(1) so that the display is only made during the first pass through the process in which it is located. DISPLAY_CONSTRUCT(string) is used to define automatic formatting of the string "string".

A number of variables with a common definition are required:

1. INPUT_LEN	input length of response
2. INPUT_WORDS	number of words in response
3. REPORT	(a binary value)
4. REPORT_1	(a binary vector)
5. REPORT_2	(a binary vector with delimiters)

Additional variables related to the number of ones in the binary vectors would also be useful as this would allow for assessing the closeness of a response to a standard.

10.15 An Example

STANDARD(ID1)[(Edmonton Alberta)NOT(Calgary)]

```
STANDARD(ID2)[(Red Deer) OR (Medicine Hat)OR(Lethbridge)]

STANDARD(ID3)[help or explain or ?]
```

Consider the student response "It is Calgary, Alta". The first step the author might want to take is to identify whether a null response is present.

```
1. PROCESS_A is
        1. EVALUATE(INPUT_LEN ~= 0/PROCESS_B) if input is not null do
           PROCESS_B
        2. DISPLAY(1) give msg first, then
        3. DISPLAY(2) this one next,and
        4. RETRY retry for response
        5. DISPLAY(3) all else fails
        6. EXIT_ANSWER exit completely

2. PROCESS_B is
        1. TRANSFORM(punctuation,response) remove punctuation
        2. TRANSFORM(remove_word_list) remove "it is"
        3. PROCESS_C

3. PROCESS_C is
        1. MATCH(keyword,ID1,match_id) match to standard ID1
        2. EVALUATE(REPORT=0/PROCESS_1) if REPORT=0 do PROCESS_1
        3. DISPLAY else display author reply
        4. EXIT_ANSWER

4. PROCESS_1 is
        1. TRANSFORM(subst(Alberta,Alta),response) subst "Alberta" for "Alta"
        2. EVALUATE(REPORT=0/PROCESS_2) if substitution not required go to
           PROCESS_2
        3. MATCH(keyword,ID1,match_id) else
        4. EVALUATE(REPORT=0/PROCESS_2) evaluate after substitution
        5. DISPLAY
        6. arithmetic etc
        7. EXIT_ANSWER

5. PROCESS_2 is
        1. MATCH(keyword,ID1,match_id)
        2. EVALUATE(REPORT_1(SUM)=0/PROCESS_3) if no word matches go to
           PROCESS_3
        3. DIAGNOSE_1(words_ok,words_miss,words_to_exclude)
        4. DISPLAY_CONSTRUCT(words_ok)
        5. RETRY

6. PROCESS_3 is
        1. MATCH_REV(keyword,id1,match_id) (match standard to response)
        2. EVALUATE(REPORT_1=(SUM=0)/PROCESS_4)
        3. DIAGNOSE_1(words_ok,words_miss,words_not_rqd)
        4. DISPLAY_CONSTRUCT(words_not_rqd)
        5. RETRY

7. PROCESS_4 is
        1. MATCH(keyword,id2,match_id) looking for wrong cities
```

    2. EVALUATE(REPORT=0/PROCESS_5)
    3. DIAGNOSE_1(words_ok,words_miss,words_not_rqd)
    4. DISPLAY_CONSTRUCT(error_cities)
    5. RETRY

8. PROCESS_5 unrecognized processing
    1. DISPLAY(1)
    2. DISPLAY(2)
    3. DISPLAY(3)
    4. RETRY
    5. DISPLAY(4)
    6. EXIT_ANSWER

The problem of obtaining from the author in an unambiguous manner and with minimum error, in terms of the processes for analyzing a student's response, the necessary information to precisely define the analysis procedure is by no means a simple matter. The problems suggested in the examples are not obviated by the use of intelligent CAI (ICAI) techniques as they generally do not address the problem with any degree of precision. In some respects the problems illustrated are more closely allied to the building of an expert system for analysis of input, but required for the analysis of each query presented to the student.

10.16 Problems Related To REPORT And DIAGNOSE

10.16.1 Report Issues

The nature of the three levels of REPORT is a function of the match procedure. For example, consider the following standard:

STANDARD(ID1)[(Edmonton Alberta)NOT(Calgary)], and input

Edminten Alta

and a character by character match producing

Report_2: 11101101 1100000

Report_1: 0 0

Report: 0

If a key letter standard is set

STANDARD(ID2)[(dmntn lbrt)NOT(clgry)]

Report_2: 11111 1000

Report_1: 1 0

Report: 0

In the first case, Report_1 gives no indication of how close Edminten was to Edmonton. This can only be obtained from Report_2. In the case of the second example, the first word may be acceptable to the author (although not correct), but we would not be able to tell the student what was wrong unless edmnt had been obtained by a transformation of the standard by removing vowels from Edmonton and we were able to relate to this transform. Alternatively, the standard Edm*nt*n, where * represents any character, could be used, but again it would not tell us what the correct characters should be where * is present. Another solution is to mark optional characters rather than use asterisks, eg., Edm{o}nt{o}n. In this case REPORT_2 for Edminten would be 111{0}11{0}1 and further processing of this string could optionally take into account the {0}s. For example, for an exact match {0} or {1} would be taken into account. For a key-letter match 0 and 1 in brackets would be ignored. Which is used should be defined in the EVALUATE procedure, e.g., EVALUATE_EXACT (recognize 1 and 0 in {}) EVALUATE_APPROX (ignore 1 and 0 in {}). REPORT would also be changed. In the case of 111{0}11{0}1 for EVALUATE_APPROX the REPORT would be 1 (an acceptable match) while for EVALUATE_EXACT it would be 0.

10.16.2 Diagnose Issues

A number of examples will serve to illustrate some of the issues related to carrying out a diagnosis of the student's input.

10.16.2.1 Example 1

STANDARD(ID1)[edm{o}nt{o}n]

response: edminten

REPORT_2: 111{0}11{0}1

REPORT_1: 1

REPORT: 1

EVALUATE_EXACT(REPORT_2 would be 0)

EVALUATE_APPROX(REPORT_2 would be 1)

MASK(REPORT_2 (would produce edm_nt_n)

MARK(REPORT_2 (would produce edmonton)

Now consider length differences as in the following two examples.

STANDARD(ID1)[edm{o}nt{o}n] ....(1)

response edm i nt o o n ....(2)

Note that the exact length of the standard is 8, the length in the standard less the optional characters is 6, and the response length is 9.

REPORT_2: 111{0}1111 and has length 8.

The length differences indicated more than the required number of characters. Similarly, a lesser number of characters can be identified. Since report is with respect to the standard, we cannot identify the extra character(s) through use of REPORT_2.

10.16.2.2 Example 2

In this example, it is suggested that the EVALUATE process work only on REPORT_2 which is then used to produce REPORT_1 and REPORT.

STANDARD(ID1)[(edm{o}nt{o}n {a}lb{e}rta)NOT(c{a}lg{a}ry)]

response: edminten elbirta

MATCH(........

REPORT_2 is 111{0}11{0}1 {0}11{0}111

EVALUATE_EXACT would produce 0 0 for REPORT_1 and 0 for REPORT, while EVALUATE_APPROX would produce 1 1 for REPORT_2 and 1 for REPORT.

If MASK were used with REPORT_2 we would get edm_nt_n _lb_erta, and if MARK were used we would get edmonton alberta with a ^ under o and o in edmonton, and under the first a and e of alberta.

10.16.2.3 Example 3

In this example consider the following:

STANDARD(ID1)[y=3+5x+8x\r2] where \r2 is reverse index.

response: y=3+4x+6x\i3 where \i is index

MATCH(.....

REPORT_2 is 11110110100

REPORT_1 is 0

REPORT is 0

MASK(REPORT_2 is y=3+_x+_x_ with the last underline raised

MARK(REPORT_2 is y=3+5x+6x\r2 with a ^ under 5, 6 and 2

10.16.2.4 Example 4

In this example consider y=3+2x+6x\r2 but a response y=6x\r2\i+2x+3 (sequence reversed). Without algebraic evaluation we would not be able to sense that the correct answer came in, unless a transform was used to first produce word units i.e., +3 +2x +6x\r2 and the same transform applied to the standard, and a matching procedure was independent of word order. Assume that a match has failed on treating the answer as just another character string. We then proceed to an algebraic evaluation:

```
        TRANSFORM(evaluate,x=1,x=2,x=3)

        EVALUATE(REPORT=0/PROCESS_A)

transformed answer is 11 31 63 as a string

        REPORT_2 is 11 11 11

        REPORT_1 is 1 1 1

        REPORT is 1 or a match

        EXIT


        PROCESS_A

        STANDARD(ID2)[(11.8 to 12.2) (30.8 to 31.2) (62 to 64)]

        MATCH(limit,match_id)

        REPORT_2 is 1 1 1

        REPORT_1 is 1

        REPORT is 1 or a match
```

NOTE:

1. The author cannot say much about an error under algebraic evaluation, for example, what could the author say if under PROCESS_A, REPORT_2 was 1 0 1. The author would not be sure; he might assume that unless a special case was encountered the answer was wrong but he could not say what in the student's answer was wrong.
2. The author could identify symbols in error under string evaluation.
3. The transform might not be done if the symbol z appeared in the answer.
4. Consideration also needs to be given to word order.

10.17 Analyzing Algebraic Expressions

Three methods for evaluating algebraic expressions can be identified:

1. Spatial; the expression is restructured into a matrix in which each row contains either superscripts or subscripts.

2. Expand to Word boundary; "words" are formed by breaking the expression into string units according to boundaries. Boundaries might be defined as the arithmetic operators and the = symbol.

3. Evaluation; the expression is evaluated for selected values, e.g., x=1,x=3,x=7.

Each has certain specific procedures, advantages and disadvantages. In some cases, an ordered combination of the three procedures seems reasonable.

10.17.1 Spatial Analysis

In this method a matrix is generated having elements which spatially separate the symbols of the expression. Consider the following example:

$$s = \Sigma(x_i - A)^2$$

which would produce the following matrix:

									2
S	=	Σ	(	x		-	A	)	
					i				

Figure 10.2 Example of Spatial Representation

The transformation TRANSFORM(spatial.....) would be applied to both the incoming answer and to the standard. A match would be requested which would compare the elements of the matrix generated from standard and from the response generating REPORT_2 as a matrix of 1 and 0 (match/nomatch). One problem which is immediately noted is that the two matrices may be of different order depending upon the number of symbols in the standard and response. Conformability of matrix order could be attained by either increasing the order of the standard matrix or the response matrix. The matrix of 1s and 0s based on the standard could be used as a mask to identify correct elements. In order to provide some form of diagnosis, the authoring system would have to obtain from the author sufficient information to create some internal representation of the element by element checks to be made. For example sufficient information would be required to define which matrix and element is to be checked.

10.17.2 Expand To Word Boundary

By expanding to word boundaries we mean that a defined set of symbols would be used to bring the expression into "words" as defined in more normal string matching of characters. The defined set of symbols would be the arithmetic operators +,-,/, and =, although others might be used eg., (and). For example the expression y=3+2x-4z would be broken into y = +3 +2x -4z, and analyzed as a set of word units.The leading element following the = symbol would have to have a + added as the standard may be in different order, e.g., y= 2x-4z+3 and broken to +2x -4z +3. The advantage of this procedure would be that those procedures used for analysis of word units could be used for expressions, with the same degree of feed back possible.

10.17.3 Evaluation Of Expression

In this procedure an expression such as y=3x+8 is evaluated for, say, x=1,x=3, and x=7 and checked against the same evaluation for the standard. The primary disadvantage of this method is that little can be told the student if the answer is wrong. It is likely to work well in some domains of subject matter, but not in others, e.g., for example when an integration symbol is in the answer.

The symbol used for the variable on the left and right hand side of the equal symbol would be important, e.g., y=3+2x vs x=3+2y the variable trial values in the first case would have to be in terms of x, while in the second case in terms of y. In some cases where an expression for a specific value may go to infinity, eg., 1/(1-X) for x=1, a default with a special flag may be required before proceeding to evaluate the expression for other values.

10.18 Reviewing The Problem Of Analyzing Typed Student Responses

The brief definition of the procedures which might be used to analyze a string response given by the student can be summarized with the following steps which the author would have to be concerned with:

1. Definition of Standard: one or more standards would have to be defined by the author; the standards would have to be identified or labeled; the definition could include the use of logical operators as well as identification of optional characters; a standard may be a string of characters or numerical values

2. Invocation of Transformations: Transformations such as "clean-up" transformations could be automatically invoked, while others would have to be specifically called for by the author

3. Matching Procedures: the author would have to define the type of matching procedures which would be used in comparing the standard and the student's input

4. Evaluation of Reports: Information provided by the transformations and matching procedures at the overall, word, and character level need to be evaluated

5. Decision Making: Once a report has been evaluated the author needs to specify what action is to be taken; the actions may include changes in flow of the course and/or a decision to display information.

The steps noted above may have to be repeated a number of times at any single point at which the student provides a response to the courseware. There may be thousands of such points in a non-trivial course. The load placed upon the author to keep track of the procedures he invokes clearly requires computer support. Such support may include "templating" a set of particular approaches to response analysis, the use of menus and default conditions, and visual representation of the procedures which have been created. Since the process is complex, it is recommended that the author be able to test immediately after defining a sequence of steps the effectiveness of the sequence. If this is possible the author can immediately decide the class of responses which may not be effectively accommodated. The use of expert system techniques would clearly benefit the author.

10.19 Evaluating The Pointing Response

Generally, the pointing response is much simpler for the author to handle. Concentric rectangles defining the target areas are easily defined by the author, and can be used to encompass any desirable degree of tolerance relative to the resolution available for the pointing process. A problem can arise when the target area which the author wishes to identify is not entirely defined by the boundaries of the rectangle. For example, if two concentric squares are displayed, and the target area is the space bounded between the squares, then the evaluation of the student's point will require a boolean representation, i.e., the point must be within the boundaries of the outer box, but not within the boundaries of the inner box. Thus, the author will require a method for specifying boolean relationships involving bounded areas of the screen. Additional problems arise when the target area is not a rectangle, e.g., a circle a triangle, or some other closed figure. Although such target areas may be defined by ORing a sequence of smaller rectangular areas, this is an obtuse method of defining the target area. Provision for defining the target area other than through the use of rectangles may also be required.

The definition of the tolerance which is acceptable to the author may be complicated when the such devices as touch sensitive screens are placed over the display monitor. In such instances calibration is required. In addition, any non-linearity of the screen resulting in distortion of the screen coordinate space cannot be easily compensated by the definition of an acceptable tolerance level in the pointing response. The use of a light pen, cross hairs, or cursor positioning by the use of special function keys or by using a mouse does not present this problem.

10.20 Specifying The Standard

Since the target area can be represented by a finite area defined on the screen, it is recommended that the direct definition of this area be the method by which the standard is defined. The definition of successive boundaries, beginning with the inside boundary, can be used to define increasing levels of tolerances for the target area.

10.21 Response Analysis Related Variables

The authoring system should contain a set of variables which are automatically used for aggregating results from answer analysis procedures. The variables usually expected by the author include the number of correct and incorrect responses. Associated with these variables would be the domain within which they would operate, and at least some elementary arithmetic operations.

CHAPTER 11

ARITHMETIC OPERATIONS

11.1 Introduction

Arithmetic operations are required as part of the courseware being created, as well as for some operations within the authoring system. Courseware needs include the use of arithmetic operations such as might be required

1. when content is being generated, e.g., in an arithmetic drill,
2. for modifying courseware flow, e.g., calculations are made as to where the student will be taken next in the curriculum content,
3. and reporting or monitoring functions, eg., score keeping.

Associated with these needs are arithmetic as well as relational and logical operations.

In the case of courseware needs and author needs special functions such as trigonometric, and matrix functions and operators may be required.

11.2 Courseware Arithmetic Needs

In some courseware the student may be required to provide an answer which must be manipulated arithmetically in order for it to be properly evaluated or transformed into another representation such as a graphical form. In drills of various kinds, frequently values have to be generated as part of the course content. The generation of random numbers is a good example, although random numbers might also be required in modeling and simulations. The author must have some method of establishing these operations within the courseware through the authoring system.

11.3 Decision Making Needs

Arithmetic, relational, and logical operations are required by the author for decision making functions. Variables defined within the student model may require manipulation by simple arithmetic operators, or by more complex arithmetic such as in making a regression estimate of future success.

11.4 Authoring Needs

The author may require access to arithmetic processes in order to create the required screen displays. For example, the generation of a normal curve or some other mathematical function may be required with the final graphical representation being the object which will be displayed to the student. Such objects may also require further manipulation by the author, such as rotation, before they are "snapped" for incorporation into the courseware by the author.

11.5 Limitation Of Arithmetic Capabilities

Clearly, there are limitations in the power of the arithmetic operations which should be made part of the authoring system. There is likely a point at which it is wiser to employ other systems for arithmetic calculations than to attempt the creation of one within the author system. For example, where extensive arithmetic operations are need by the author, it may be preferable to call upon a regular Fortran compiler, or an APL interpreter and interface such software to the author if necessary.

11.6 Arithmetic Variable Types And Forms

Real, integer and string variables are required for use by the author. These may take the form of scalars, vectors or matrices. Provisions should be available for the use of matrices having character elements as these may be required in answer judging procedures. Although binary switches may be used, from the author's point of view, these may appear as simply integer variables.

11.7 Arithmetic Operations

The usual arithmetic operations on scalars should be available, i.e., add, subtract, multiply, divide, and exponentiation, as well as the relational operations. If matrices of scalar elements is available, then the inversion operator should be provided. The order of precedence should be that used in most other numerical type software, and the parentheses used as indicators of grouping operations. Assignment may be the equal symbol or a left pointing arrow. Conversion of types during computation should be as used in Fortran. If the authoring system is required to provide extensive computation capabilities, then consideration should be given to accessing an APL interpreter.

11.8 String Operations

A number of string operations are required for the answer judging algorithms used by the authoring and the run-time system. In such instances the author is not likely to use the operations directly. However, if the author is going to be provided with flexibility in templating a unique answer judging function, then some string manipulations will be required:

1. assignment as in arithmetic operations
2. concatenation or joining of strings
3. convert string to vector or matrix form and related indexing
4. masking operations
5. string compression and expansion
6. equality of strings
7. sorting of word strings
8. creation and manipulations of lists
9. digit string to real or integer conversion
10. real and integer conversion to string digits

11.9 Special Short Form Operations

For sake of efficiency, and for ease of use by naive authors some arithmetic operations should not require the definition of the usual mathematical statements. For example, where score keeping is required, the author should be able to simply set a switch to on and define the conditions for which a count of 1 is to be made.

11.10 Naming Convention

There should be no special naming convention such as used in Fortran, i.e., variable names beginning with the character i,j,k,l,m, or n are integers. Names will require a length restriction, and may not begin with a blank character or a digit. System reserved names, including sin, cos, tan, etc., are usually not available as variable names. In addition, those system variable names which may be automatically defined for score keeping or for screen control e.g., last cursor location, may also be restricted in that they cannot be defined by the author. They should, however, be available for use by the author.

Because a large number of variables may be defined by the author for some courseware, the authoring system must be able to provide support to the author in recalling the variables already named, their type, and where appropriate their size. For example, the author should be able to have a list of currently defined variables displayed. Upon option, the author should be able to also have displayed the type of variable for each name displayed, its size, and when applicable, change the value or contents of the variable. These features are also likely to be required for debug purposes.

Chapter 12

Interfacing the Author to the Authoring System

12.0 Introduction

Although the authoring system may be constructed for direct entry, consideration should be given to making the authoring system part of the CAI system so that those features available to an instructor and student are also available to the author. These features likely include the registration system as well as the run-time system used by the instructor and the student. It is very useful for an author to debug new courseware by ensuring that it operates within the environments used by the instructor as well as the student.

12.1 Author Entry to the CAI System

Since the author should have access to executing the courseware as it would be seen by an instructor and a student, one of the menus should allow the options to operate as an author, an instructor, or as a student. The CAI system should distinguish between the special case when an author is operating as an instructor or a student from the case in which an instructor or student is operating, as there are likely to be special provisions made for the author, e.g.,, not saving a restart record and access to parameters used in the courseware. More specifically, the following environments are likely required:

1) an author operating under the authoring environment as an author, as an instructor, or as a student

2) an author operating within the instructor's environment

3) an author operating within the student's environment

Consideration needs to be given to the hierarchical nature of the users of the CAI system. Our recommendation is the following hierarchy:

1) Manager and operators

2) Authors

3) student authors (if applicable)

4) instructors

5) students

Managers and/or operators are frequently required to intervene on behalf of a student to carry out special actions. For example, a student might accidentally power off a terminal, destroying a set of special character fonts and require special action for their reloading. Of course, the physical environment in which the student operates may dictate to whom special actions are allowed.

Student authors present special problems in that all privileges provided the usual author may not be assigned. Thus within the category of authors, provision should be made to define an author privilege level. The privilege level may be used to determine, for example, what type of authoring processes are available, and whether more than one course can be authored under the same author ID.

Instructors are likely to be given access to special processes for monitoring student progress, as well interacting with students. These processes may be required to interact with the courseware in a special manner. Thus the author will be required to test the adequacy of such interaction as part of testing the execution of a course. For example, a test score may be stored for retrieval by the instructor as part of the instructor's role of monitoring student progress.

The student environment is likely to be the least flexible in the context of the authoring system users. For example, it is not likely that the student would be allowed to behave as an author with privileges to modify the courseware being taken. In this context special consideration must be given to special keyboard actions which may be available to all users and which may abort a process, or return the user to the operating system. A student with access to operating system commands could cause special problems, such as the creation of files and unauthorized use of the system facilities.

12.2 Author Entry to the Authoring System

Initial entry into the authoring system should make provision for the following:

1) display of the registration parameters including the names of courses which the author can author

2) a recall of any author notes created upon leaving the last authoring session

3) presentation of the authoring parameters currently invoked for the author, and where permissible, allow the author to change these parameters; some parameters for authors having the higher privileges might detail the types of editors and interpreters which they may invoke.

4) information which will act in a confirmatory manner supporting the author's intentions for the current authoring session.

5) the display of authoring statistics which may include such parameters as accumulated authoring time for the specific courseware to be authored, amount of courseware generated, number of drawings generated, number of versions, etc. These can also act in a confirmatory manner for the author.

6) special consideration must be given to an author entering the system for the creation of a new course, in particular documentation should be mandatory for non-test courseware. (Non-test courseware is courseware which an author may quickly create to test an idea, but not likely to be saved beyond the current session).

Assuming that the courseware is not trivial, the author should be provided with information which can be used to confirm that entry is being made to the desired section of the course, and those authoring parameters which can be adjusted for the current session. Where appropriate, the logical consistency of the parameters which may be set by the author should be checked. For example, if during a proposed session new courseware is to be appended, then courseware must already exist.

12.3 General Author Interfacing Considerations

A primary goal of an acceptable interface is to minimize errors due to memory load, and incorrect moves made by the author. Approximating this goal requires consideration of how information is presented to the author, in what sequence, and how adequately is the author's actions checked for validity. In addition, the ease with which information can be stored and presented to the author is intimately related to the software architecture of the authoring system.

The following are recommendations found useful to assist the author and which are not directly tied to architectural considerations of the software:

1) The author should never encounter a screen display which does not also contain a menu of available commands, or a note concerning the current action or available actions.

2) When a menu or set of commands is available, only those which are valid for the current context should be displayed.

3) The author should not have to search through a series of menus to locate the desired process.

4) Where an action by the author does not create an observable screen effect, confirmation that the action was completed or rejected should be made. Frequently this can be accommodated through simply displaying the word "done".

5) Important options which may apply to a series of actions should be visually maintained for the author over the sequence of actions. For example if the author is creating a template which involves many instructional moves, a signal should indicate that this mode has been enabled.

6) Context dependent help should be provided on demand by the author. For example, the display of a special font set may be required during the time an equation is being created.

7) Where possible, the last four or five moves of the author should be saved for possible display at the option of the author. An interruption to the authoring process would require the author to recall the last sequence of actions, and this sequence may not be obvious from the content of the current screen.

8) The author should be able to call for display of information pertinent to a particular action.

9) As far as possible, consistency of meaning should be maintained in the use of commands, icons, and special function key actions. For example, commands such as quit, stop, exit should be used with consistency.

10) Complex actions may require graphical representations in order to be fully and easily comprehended by the author. For example, the transformation of a student's input prior to analyzing the response can be complex and could be more easily understood when supported with a visual representation of the transformations being used.

11) As far as possible, the author should be able to retreat from an action. For example, for the dichotomous choice of calling for a student input either from the keyboard or from a pointing action, the author should be able to retreat from authoring a pointing response when a keyboard response was intended.

12) Specific information related to the strategies created and the curriculum content authored, should be automatically generated and made available at the option of the author. For example, at the beginning or end of an authoring session a course map of the logic and content is useful.

13) Where appropriate, the author should be able to "try out" a move before making a commitment to the action; for example, an author should be allowed to display a drawing in the context of creating a screen display without making a commitment to its use.

14) The interface should provide for minimizing the memory load placed upon the author. For example, elements named by the author or by the system should be accompanied by author defined notes and placed into a list.

15) A help option should always be available. This option may explain the action of a menu item or other features such as recovery from unintended actions.

16) When a subsystem can be entered from different authoring sequences, the calling system sequence should be identifiable by the author. For example the subsystem for placing text on the screen may be entered as part of an answer sequence when a reply is being constructed for the student, as well as from a main menu. The author should know from which subsystem the text definition mode was called.

17) The definition of special function keys should be easily retrievable by the author.

18) Processes which once initiated and not easily reversible should require confirmation before being carried out.

19) Upon initial entry by the author into a subsystem, the option to back out of the subsystem should be available. This is required in the case the author has entered the wrong subsystem.

20) Errors should be identified as close as possible to the offending input. Where parsing of a set of parameters is involved, then the point at which parsing failed should be made available to the author. If system action was made using unoffending parameters, those which were used should be identified or the point of failure identified.

21) Author input by typing of parameters and names of objects should minimize the chance of errors. For example, when a list of variables is presented to the author the list should be indexed, and when appropriate, the index used as the identifier of an element in the list. There are situations where this may not be appropriate, as for example, when the use of the listed items are used to construct an arithmetic statement. Keyword skeletons are, however, better handled by the author referencing them by an index.

22) Since many authoring sessions are likely required to create a non-trivial course, the author should be able to execute that part of the course immediately preceding

the point at which new material is to be added. This provides for continuity of style and content.

12.4 Interfacing the Author to the Course Structure

The nature of the author interface to the course structure will depend upon how the instructional units are structured, e.g., they may be structured hierarchically, as well as upon the internal representation of the course structure. The course structure is likely to involve units of an instructional and curricular nature, peripheral objects integral to the instructional units, data components, student related components, and controlling components.

Generic units which an author is likely to use might involve one or more of the following:

1) Curricular Units: course, section, chapter, lesson, etc.

2) Instructional Units: review, diagnostic and competency assessing tests, introduction, tests, pre-organizers, simulations, drills, tutorials

It is through the use of these units that an instructor will maintain a conceptual structuring of the course being created. The curricular units are likely to have an overall hierarchical organization, while instructional units are likely to be less structured as their use is more dependent upon the author's perception of how student's learn.

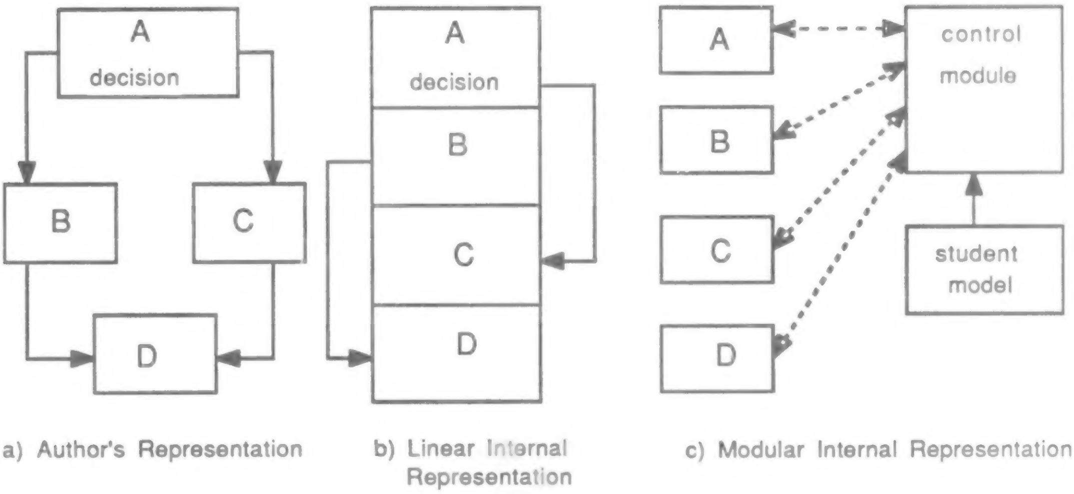
12.5 Interface to Curricular Units

The author requires some vehicle for naming and structuring the curricular units during the authoring and debugging process. Once these units have been created, the author must be provided with access to a representation of the structure. This access is required in order to maintain continuity in the authoring process and to provide entry to specific units for purposes of modification and testing. There is merit in curricular units maintaining a link to the corresponding instructional units which it contains. For example, an author may label a curricular unit as a lesson with the qualification that it is a lesson on the topic of solving two simultaneous equations.

Ideally the author should be able to obtain together a representation of the curricular and instructional nature of any part of the course, and the logic which is associated with it. The addition of the logic component adds another dimension to the representation required, for example, it may require reference to a student model component. The author should also be able to trace a specific student's path through the course in terms of the representation.

12.5.1 Micro View of Course Structure

The capability of the authoring system being able to provide a meaningful representation of the course structure for an author does depend upon the internal organization of the course, i.e., how the data structures holding the information are organized.



a) Author's Representation b) Linear Internal Representation c) Modular Internal Representation

Figure 12.1 Different Representation of Course Structure

With reference to Figure 12.1, a simple hierarchical representation is given for course units A, B, C, and D. The author's representation indicates that the starting point is A, followed by a decision to invoke B or C, and then D. Figure 12.1(b) indicates one internal course representation which is strictly linear. If the author's representation is to be built from the diagram of 12.1(b) the complete sequence of code forming the internal representation would have to be read in order to the build the author's representation. In addition, any change to the internal linear code, such as an addition of further code, would require a complete re-read in order to build the author's representation.

In the modular representation illustrated by Figure 12.1(c), the control module contains sufficient information for creating the author's representation. Changes to the modular units, A,B,C, or D, since they do not contain decision making capability, would not require a rebuilding of the author's representation. (Modular units may have decision making capability, however, their representation would likely be made at a micro level and for different purposes than representing the curricular structure and content.) However, if a student model is part of the course structure, its contents and contribution to the decisions of the control module would have to be determined in order to provide the author with a complete representation.

12.5.2 Macro View of the Course Structure

It cannot be assumed that the optimal representation of the course structure is best conveyed through a diagram as illustrated in Figure 12(a). Consideration needs to be given to a macro and micro view of the structure, as well as non-iconic representations. A non-trivial course requiring one hundred or more hours to complete, would likely require a very complex tree structure which would occupy considerable area on the surface of a display monitor. Thus it should be possible to have iconic representations at different hierarchical levels of the structure as illustrated in Figure 12.2, as well as a textual descriptions.

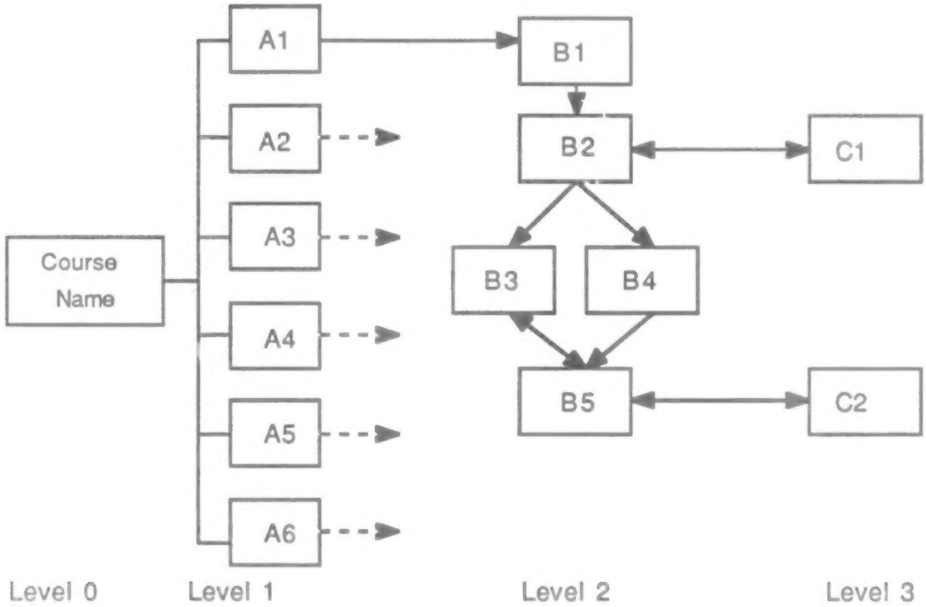


Figure 12.2 Diagrammatic Representation of Course Structure

If a diagrammatic representation is provided for the author, then the author should also be provided with the option to determine the extensiveness of the levels being represented. Since during the testing phase of a course, frequent access to editing may be required at a level deeply embedded, e.g., Level 8, the path to access the exact point at Level 8 should be retained for purposes of repeated direct access.

Since a diagrammatic representation could be complex and require considerable space, or alternatively a micro view, the presentation of a text-based representation such as illustrated below should also be possible.

```
stat2               Descriptive Statistics
                          Central Tendency
                                mode
                                median
                                mean
                          Variability
                                range
                                standard deviation

                    The Normal Curve
                          etc
```

Not only does the above representation illustrate the course structure through indents, it also conveys the content of each unit within a level. The conditional nature of presenting a topic, is however, not easily illustrated.

Regardless of which type of representation is used, the author should be able, for example by pointing, to "open" up information about the unit, or to invoke its editing or execution.

12.6 An Integrated System of Information about the Course

A framework of data structures is required to structure the representation of a course in order to provide the macro and micro view of the content units and general instructional strategy employed. However, this structure should also relate to other types of information which is required by the author. An expansion of Figure 12.3 is given below to illustrate the additional type of linkages to other information which is required.

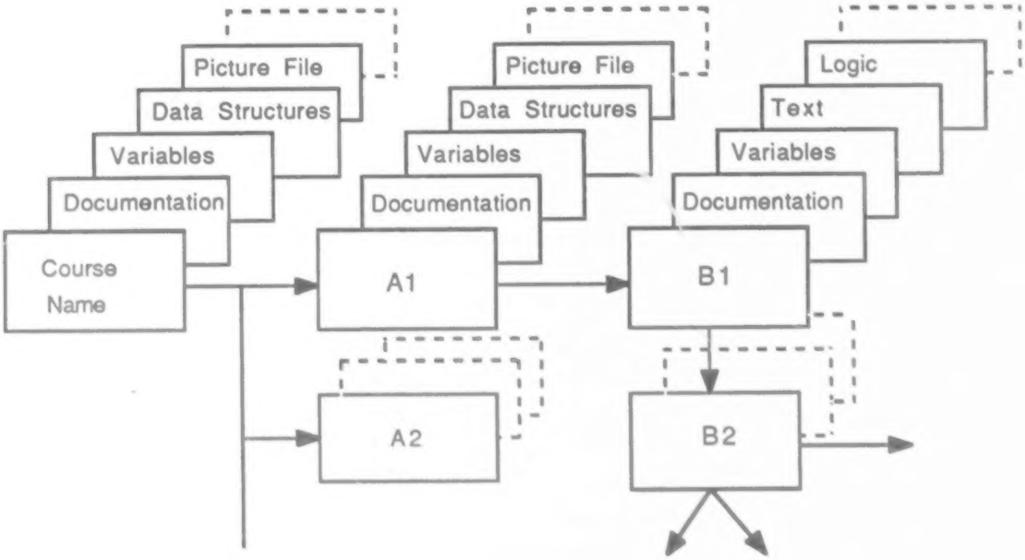


Figure 12.3 Expanded Representation of Course Structure
with References to Associated Information

Figure 12.3 illustrates how the original structure, illustrated in Figure 12.2, might require expansion in order to provide the author with access to supporting information of each content unit. This type of structure would likely be created through the tree structure building facilities of the programming language in which the authoring system is built. Such structures as illustrated need to be created and destroyed dynamically, and must be planned for early in the system design process.

The supporting information in Figure 12.3 represented by boxes in the background are only examples of the kind of information which may have to be stored.

In some instances, the author may wish to recreate the passage of particular student through the course by accessing for input, elements of the student's performance record. In such instances a further linkage is required to external files. Such an external file may be a pseudo student file which the author may use to ensure a specific pathway through the course. The latter can be important for testing the adequacy of decision making within the course. Similarly, linkages may be required to data which the course requires for the presentation of course content.

Chapter 13

Defining and Interrelating Functional Units

13.0 General Considerations

Although there are functional units which will have general applicability regardless of the subject matter content of a course being created, the power and versatility of an authoring system is frequently judged by its capability to provide specific features for the author. For example, a feature which provides a grade three teacher with the ability to easily create instruction for subtraction, will not be viewed as an important feature for an instructor of university organic chemistry. Such features are best provided by having the capability to easily tailor an authoring system to the specific needs of the author.

Another modulating characteristic in the design of an authoring system will be the capabilities of the authoring terminal as well as the student terminal. An authoring terminal having, for example, capabilities for 3 dimensional diagrams and rotations will require more authoring options than one which does not have a graphic capability; student terminals which have access to video disk and audio, will require authoring functions to control these devices.

The creation of short sequences of instruction, i.e., lessonware, in contrast to lengthy sequences which may take hundreds of student hours to complete, will require different capabilities for creation and debugging as well as providing for a proper restart position for the student at each session following the first.

Where time consuming sequences of instruction are created, the accidental loss of courseware due to system errors suggests that in some environments recovery procedures or alternate procedures for interfacing to the courseware is required. One alternate procedure is to be able to escape to the computer's operating system in order to use system utilities. Some micro computer authoring system do not provide for this capability.

13.1 Veridicality

The authoring system, in addition to providing an adequate interface for the author should also represent to the author as closely as is possible exactly what the student will see during execution of the course. There are of course limits to this capability because not all actions taken by the author will be directly observable to the student, e.g., arithmetic operations, algorithms for answer judging, etc. The processes of algorithms used for answer judging should, at least by option, be made observable to the author.

13.2 Functional Units of the Authoring System

13.2.1 Entry to the Authoring System

In designing the authoring system consideration needs to be given to intializing the system upon first entry and subsequent entries of the author. In some instances consideration must be given to interfacing to the user registration system provided by the computer's operating system.

Factors which should be considered in the design of the software providing the author with entry into the authoring system are as follows:

1) Some features of the CAI run-time system will be required by the author. Therefore consideration should be given to have the author's initial entry made through the facilities of the CAI system.

2) Some parameters of the authoring system may be considered fixed or adjustable by the author depending upon the privilege level set at the time an author is registered on the system. In a research environment, for example, the author may wish to have the option to invoke a special interpreter or editor for a particular session, or to invoke a complete procedure over the content of the course, e.g., re-build a course map which normally would be created dynamically.

3) Provision should be made for the author to recall "authoring notes" left during the last session. In some environments, such as in a public institution, the instructor's primary task may not be that of creating courseware, thus there may be lengthy periods between authoring sessions. The recall of notes would permit the author to be reminded of tasks still to be performed and better provide continuity from one session to another.

4) Some files will have to be initialized or opened (transparently for most authors). Consideration must be given to error messages associated with these processes, as for example, when an expected file cannot be opened or does not exist.

5) It is useful to the author to have entry to, and exit from the authoring system dated and time "stamped" and to have such information presented on entry. Files which are transparent, should be provided with such identification, even though these files are not directly observable by the author since such information is helpful for the tracking and isolating system errors.

6) Some monitoring functions of how the author makes use of the authoring system can be very helpful for isolating errors. Since it is almost impossible for any user of a computing system, regardless of how knowledgeable they are, to accurately recall a sequence of moves prior to the appearance of a system error or bug, appropriate files to hold all authoring actions need to be opened and date-time stamped.

7) Since many authoring sessions are likely to be required for the development of any major courseware, files which provide continuity from one session to another must be considered. These files would contains lists of names for variables, objects, or stored strings. For example, if a complicated formula has been stored with a name in a list for repeated use, the author will want access to it at some later authoring session. The creation, opening, writing, and closing of these files are likely to be entirely transparent to the author.

13.2.2 Starting the Authoring Process

Provision must be made for the author to identify the location in the course where authoring is to start. For lengthy courses, it is important that an author be able to get to the point at which authoring is to start with some degree of efficiency. A course map, for example, might assist with identifying the general region where authoring is to start.

The granularity of the course map and the extent to which non-screen processes of the course are represented within the course map, will determine how precisely the location can be identified.

In order for the author to re-establish the train of thoughts associated with the creation of courseware at the last session, provision should be made to allow the author to execute the course just prior to the point where authoring is to begin. This is important when objects previously displayed have screen positions which determine the positioning of new material. A useful option may be to default to the last authored position in the courseware, or to provide a list of locations which were last authored. It is likely, however, that the latter would have to be accompanied by author defined notes.

When provision is made for the author to execute the courseware, then consideration must be given to the problem of ensuring that such an execution can be made by entering directly rather than through proceeding material. This may not be a simple matter especially when the course is not frame oriented where each frame contains sufficient information for its execution, or when the internal representation of the course is not hierarchical with a precise definition of the scope of variables being used. For example, where named constants have been set before the point of execution, or when the execution of current code depends upon a preceding input, an adequate execution may not be possible.

13.2.3 General Authoring Functions

The following list of authoring functions are considered to be sufficiently general as to be required in any authoring system regardless of subject matter:

1) A major component of course development relates to the placing and formatting text to a display screen. Thus a major function of the authoring system is to allow the author to carry out this function, however, a number of sub-functions are required:

    a) screen position control for placement of text; an added feature may be the use of a formatting function rather than the formatting being defined by the author

    b) the use of superscripts and subscripts

    c) where appropriate, color changes

    d) font changes and easy access to a description of font sets

    e) definition of the number of columns and rows of the screen, and areas defined for specific special use, e.g., a reply area, a menu area.

    f) guidelines for formatting the screen, e.g., vertical and/or horizontal lines to assist the author with text and drawing alignments on the screen

    g) erasure of the screen and portions thereof

    h) the display of information stored by the run-time system, e.g., the student's name, an examination mark, etc.

2) The author must be able to identify a starting point within the course in order to carry out a number of functions. Such definition may be accomplished by the use of icons, a display of a list structure of the course, or alphanumeric names.

3) For lengthy courses, the author must be able to identify a restart point for the student.

4) Graphics are required, but these capabilities may have to be tailored to specific author needs and as a function of the content of the course. It is desirable to be able to mix textual screen displays with graphics. Graphics in the sense of drawings or diagrams, and plotting of data may be required. The establishment of a graphics library should be considered.

5) Student control over the length of time the screen will be displayed is required. Provision should be made for the student to backup one or more frames. (This may not be a simple matter if the internal course representation contains conditional "branch to" statements; a stack of pointers and variables associated with conditional branches would have to be continuously saved).

6) Storage of information is required for use during the authoring process as well as during execution of the course, e.g., the author will need to store strings which may be sets of keywords or phrases used in answer analysis algorithms.

7) The definition of student input from the keyboard or through a pointing response is required. In some specialized courseware, input may be required from a peripheral device. Input may require the definition of a time parameter and a length parameter. Consideration may need to be given to what action the run-time system will take if either parameters are exceeded.

8) Transformation of student input is is likely required before answer analysis can be carried out e.g., leading blanks may have to be removed. Some transformation functions will be related to the requirements of answer analysis algorithms

9) Answer analysis algorithms are required. A minimum set of algorithms can be defined, however it is not likely that one general set will satisfy all authors; algorithms may be subject matter specific.

10) Arithmetic operations are likely to include the basic operations required for score keeping, however, the definition of complex decision making which may be based upon a student model, may require more complex functions; plotting capabilities may also require more complex algorithms e.g., a sine function; consideration needs to be given to the creation of variables and their types; some operations may be automatic once enabled, e.g., score keeping

11) The exact nature of branching capabilities are likely to be determined by the nature of the internal course structure, which if hierarchical, may not require any explicit branches at all.

12) It is most useful for the author to annotate or to attach comments to various sections of the courseware.

13) The display of lists of information useful to the author during the authoring process is required, e.g., a display of the names of stored strings, drawing, etc.

14) Options are required for interfacing to the run time system for purposes of testing; some options may require the definition of parameters in support of debugging.

15) To increase authoring efficiency a feature is required which will allow the definition and repeated use of an instructional sequence, e.g., the capability to develop templates.

The functional units noted above should be developed with a high degree of modularity. This becomes very important when functional units require replacement because of an enhancement, and/or when for a particular author a specialized function is introduced. For example, it should be possible to easily call a specialized answer analysis algorithm.

13.2.4 Text to Screen Functions

An example of the interrelationships among functional units related to the formatting of the screen is given in Figure 13.1.

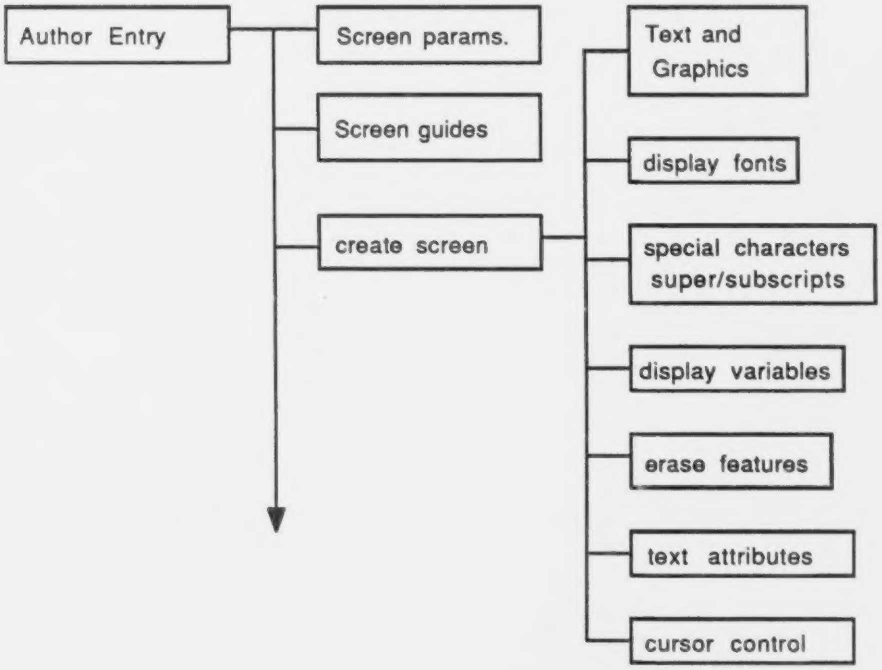

Author Entry
Screen params.
Screen guides
create screen
Text and Graphics
display fonts
special characters super/subscripts
display variables
erase features
text attributes
cursor control

Figure 13.1 Screen Related Functions

Not all features illustrated in Figure 13.1 may be required nor possible depending upon the nature of the terminal facilities used for authoring and delivery of the courseware. For example, cursor control may not be required if a fully functional visual editor is used as the basis for creating text for display.

13.2.5 Text-Graphic Functions

Figure 13.2 illustrates some of the simple graphic functions which are likely required in the context of defining a screen containing text.

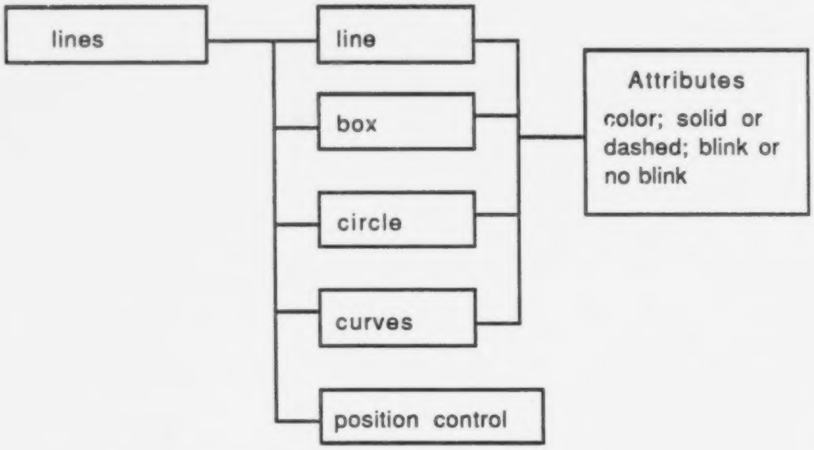


Figure 13.2 Line Drawing Functions

The functions illustrated in Figure 13.2 may be considered part of graphic functions, however, the functions noted should be available during the operation of placing text to the screen as they are useful for drawing attention to specific text, e.g., placing a box around part of an algebraic equation.

13.2.6 Graphic Functions

Graphics functions are likely to be the most complex set of functions required by the authoring system. The nature of the functions may be very closely related to the nature of the subject matter being created. The specification of shading functions can very much depend upon the nature of the terminal hardware and software which is available. Figure 13.3 illustrates some of the functions which are required.

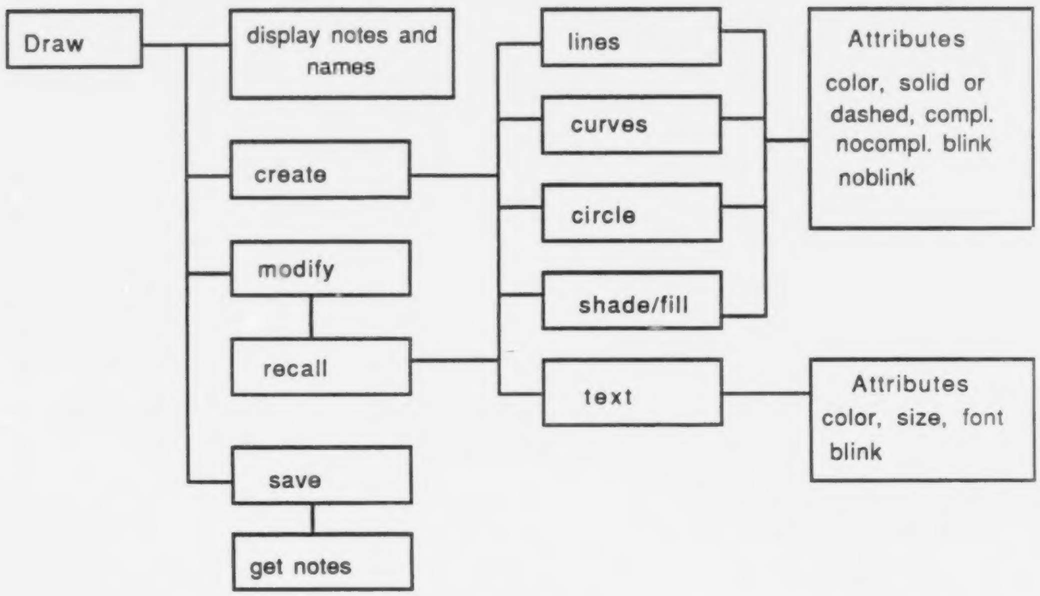

Draw
display notes and names
create
modify
recall
save
get notes
lines
curves
circle
shade/fill
text
Attributes
color, solid or dashed, compl. nocompl. blink noblink
Attributes
color, size, font blink

Figure 13.3 Draw Functional Units

The author should be provided with the capability of modifying an existing drawing, renaming it, and storing it in a library for repeated use. Additional features which could be useful are as follows:

1) rescaling the size of the drawing

2) rotation of drawings until a desirable orientation is obtained

3) "test" drawings -a try-out of a drawing without commitment to use the drawing in the courseware

Since drawings require an identification for retrieval, the software must ensure that duplicate names are not allowed.

It is not hard to envisage specialized needs of authors with the regards to graphics. For example, where an author has many electronic circuit diagrams to create it is desirable that specialized graphic functions be built. In the latter case for example, an author might wish to create an electronic circuit by "picking" components from a menu.

13.2.7 Plotting Functions

The software for authoring 2 dimensional plots for instructional purposes should be considered as different from true data plotting software. Although plots for instruction must be just as accurate, especially in shape, as plots used for representing data, the amount of detail as well as their nature are different. Many examples which illustrate such differences can be seen in good textbooks.

Plots for instructional purposes have the following characteristics:

1. Axes may or may not be plotted.
2. Unit values on the axes may be arbitrary; they may be integer, or real values, or even alphabetic characters.
3. The clarity of the concept being illustrated is of primary importance; there is less "pizzaz", less portrayal of useless dimensionality, easily readable character fonts, and only those lines which enhance understanding the concept being illustrated.
4. Instructional plots may contain explanatory notes, and "pointers" to relevant information.
5. There may be many different arrangements of the axes to accommodate the plotting of values within the domain of interest.
6. Some selected points which are plotted may have projections made to the X or Y axis, or both axes.
7. Portions of an area under a curve may be shaded; this is frequently required in probability and calculus.

The author should be able to use simple numerical values to create most plots. Such values may be entered directly from a keyboard or read from a data file. Since the plots are likely to be used in more than one place in the courseware, they should be named and placed into a graphics library with a brief description of their nature.

Where complex functions are required, e.g., normal curve, quadratic, complex exponential functions, these should be generated by other software and stored in a file for the author to use.

The author should be able to specify the axis length interactively and explicitly on the screen, and if required, to stop at this point. (Such axes may be required as a basis of a student's response). The author should be able to specify the value of each unit on the X and Y axis, or allow the software to do the scaling. (The latter is not recommended as it may result in fractional values assigned to each unit, rather than simple integers). The author should also be able to make many plots in relative address form so that a set may be moved as a unit and be relocated to any part of the display screen.

Figure 13.4 illustrates some of the main functional units of the plotting software.

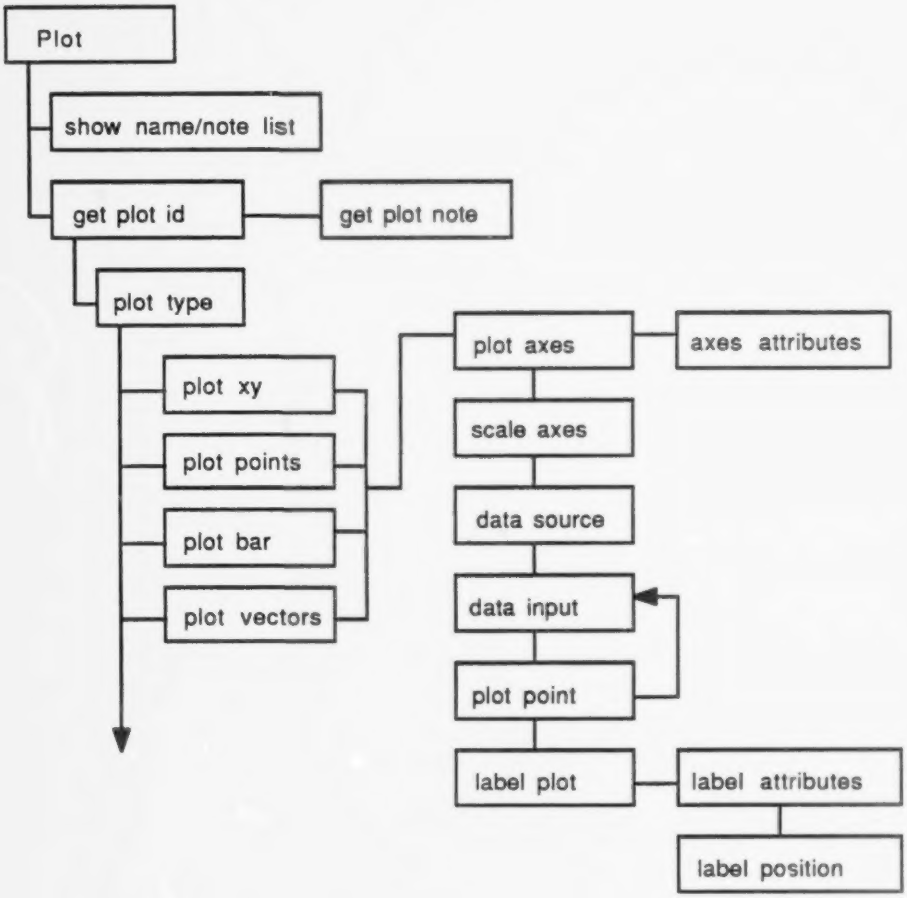

Plot
show name/note list
get plot id
get plot note
plot type
plot xy
plot points
plot bar
plot vectors
plot axes
axes attributes
scale axes
data source
data input
plot point
label plot
label attributes
label position

Figure 13.4 Plotting Functional Units

The three types of plots recommended as a basic set are as follows:

1. plot a continuous function
2. plot each data point
3. plot a bar graph
4. plot vectors

For continuous line plots break points should be available in order that attributes of the curve can be changed, and so that discontinuous sets of functions can be plotted. In the case of data point plotting the author should have the option of selecting the character to be plotted and its color. The label function should allow accurate positioning of labels for each marked unit on the axis, as well as positioning anywhere within the space defined by the plot. Font changes, superscripts and subscripts should be allowed as part of labels. It should be possible to plot more than one continuous function on the same axis.

Three dimensional plots are highly desirable, especially if rotations can be performed by the author to ensure that the student will see points from a desirable perspective.

Contouring a three dimensional graph is also highly desirable. It is likely that fairly complex software would be required for the handling of three dimensional plots.

It must be emphasized again that the type of plots noted here may be too specialized for some subject matter areas of the lower grades.

13.2.8 Displaying the Graphics

Provision is required for the author to display graphics already created, and likely retrieved from a library of graphics objects. Retrieval requires that the author identify the appropriate drawing, and this can be difficult when hundreds of drawings exist for the same course. Thus, the author should be able to at least review the drawing ids and associate notes, and if possible have the drawing displayed on the screen without making a commitment to its use in the courseware being created.

If the authoring terminal software is capable of rescaling, the option to rescale a drawing is useful as it allows "fitting" it to the space available. Alternatively, displaying a drawing on the screen and then placing text around it is a simple and adequate procedure.

Figure 13.5 illustrates some of the basic functions required for displaying a graphic.

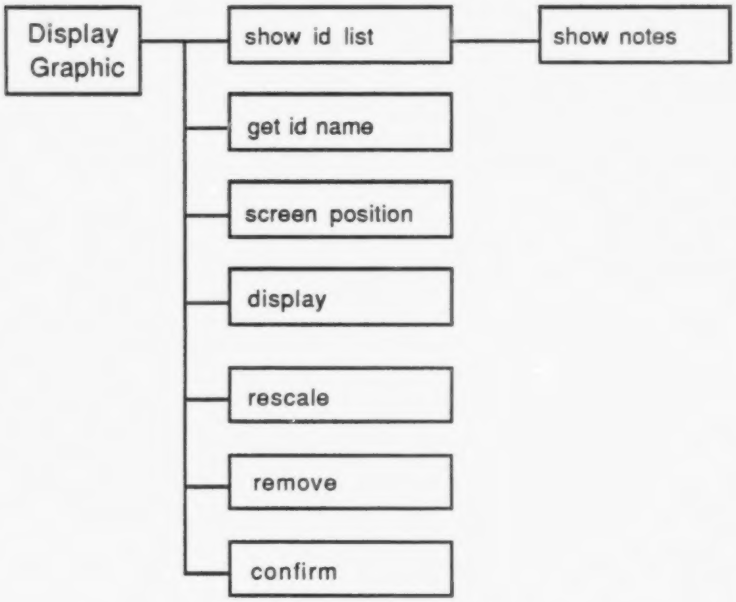


Figure 13.5 Graphic Display Functional Units

13.2.9 Screen Erasure

In most instances screen erasures are easily handled for rectangular areas of the display screen. The three basic actions required include erasure of the complete screen, erasure over a defined set of row boundaries, and a rectangular area marked on the screen by the author.

The erasures noted above will not handle the erasure of a diagonal line pointing to some other feature of the screen. Such erasures require turning off exactly those pixels which were lit. The negation of the write instruction is usually required to effect such an erasure, however, some pixels of characters over which the line was originally written will be damaged.

13.2.10 Store Functions

The author requires the capability to store and retrieve strings under an author defined name. Such storage might include keywords used frequently for answer analysis, as well as complex equations. (The latter in some instances might be better accommodated through a graphic representation). This facility is more than the usual "cut and paste" facility provided by good visual editors, since the storage is likely to be long term and not just during the current authoring session.

Where the internal representation of the course allows for the storage of strings, such as a student's name, then the author will require the capability of retrieving a string from storage and placing it in the appropriate storage for use during execution of the courseware. The fact that such a store has been made must be maintained from one authoring session to another.

Figure 13.6 illustrates the basic functional units.

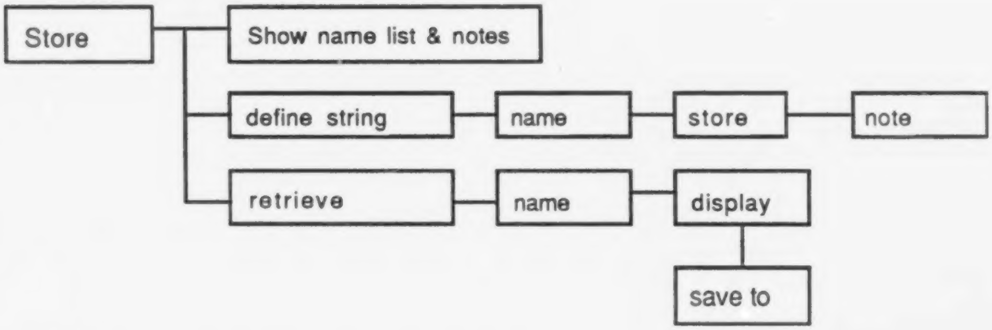


Figure 13.6 Store Functional Units

13.2.11 Input Functional Units

The capabilities of the student terminal are important to the definition of the input during the authoring of courseware. Usually a keyboard is available and most authors would want to make use of it. (In some training applications authors wish to reduce the problems of answer judging by making use of only a pointing response). The facilities available for a pointing response could be a mouse, a set of cross hairs controlled by special keys, a light pen, or a special screen overlay which allows the student to point with a finger.

In the case of input from the keyboard, the authoring system may have to allow for the following specifications by the author:

1. the exact location on the screen where student input is to be displayed
2. the length of the input, usually in characters and/or lines
3. the use of any special character fonts as well as superscripts and subscripts
4. the length of time the system is to wait for a student input before proceeding
5. the identification of the input's location in the context of the course
6. whether the input is to be treated in some special manner, e.g., is the input to be recorded as part of the performance records

In defining the length of the input, consideration must be given to which characters the run-time system will count as input. For example, when the student is to input an equation which may require superscripts, subscripts, and underlining, the number of non-character elements will be greater than the number of character elements. In this latter example, there is merit in the system only counting the alphanumeric set. In defining the space to be occupied by the input, consideration needs to be given to equations which may occupy two or three separate character line positions.

If input can be made in a special font as required in the teaching of a foreign language, then the author must be able at the time of specifying the input, to define the appropriate character font which will be displayed on the screen as the student types.

The length of time the system may wait before taking action other than waiting for input can be important in instructional situations where the objective is to increase the speed of response, e.g., a simple arithmetic drill.

An identification is required for the input defined by the author. This identification is the only "handle" which identifies the student's input with a specific location in the course. Such information is mandatory if any sense is to be made of the performance recordings left by the students. For trivial courses, or lessonware, such identifiers may be defined by the author, however, such definitions become error prone when hundreds or thousands of responses are initiated by the courseware. In the latter case a special software system should be responsible for generating the unique identifiers.

The simplest way for the author to define the location on the screen for the student's input is to define a rectangular box forming the boundaries of the input. The software of the authoring system must transform the coordinates of such a box to the rows and columns coordinates of the current screen before the rectangle is displayed. Of course, confirmation is required that the rectangle displayed is adequate.

Although not of great importance, the author may wish to set the color associated with the student's input.

The definition of input for a pointing response would require only that the author set a time parameter. This type of input should also have associated with it an identifier. A more important parameter associated with the pointing input is whether the input is to be for a defined target or is the input to be a continuous set of points as would be required if the student was asked to use the pointing device to draw a line or curve. If the author views the student environment as one in which the student might attempt to input a very large number of points, as would occur if the student "scribbled" with the input device over the screen, then the length of the input parameter may require definition by the author.

Figure 13.7 illustrates the basic functional units for defining the input.

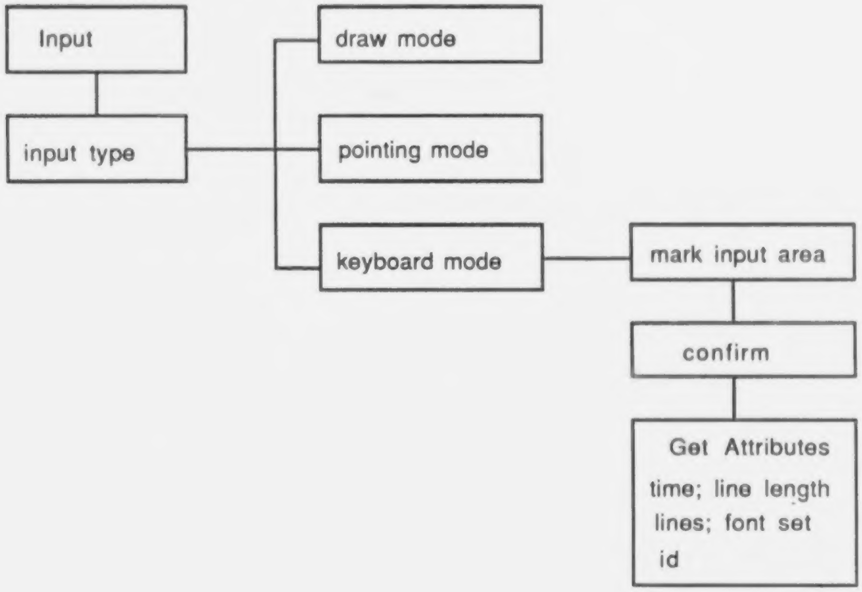


Figure 13.7 Input Functional Units

13.2.12 Input Transformations

Input transformations are defined to be those transformations which are placed upon the student's input as it is obtained from one of the input modes described above. The transformations will likely operate on strings, such as a phrase, an equation, coordinates of a line, or upon numerical input such as a number or vector. The authoring system should contain common transformations such as the following:

1. the removal of leading and trailing blanks
2. replace a string in the input with defined string
3. change case, e.g., change uppercase to lowercase
4. remove word units according to one or more defined lists of words
5. remove special characters which negate each other
6. remove vowels

In some cases, the transformations may be required before some answer judging algorithms are applied. The original student's input should be preserved before one or more transformations are applied. This is important if the author wishes to compare the input response with the transformed response. The author should be provided with a report which can be interrogated in order to determine whether the transformation was in fact applied and the input transformed.

It is likely that in some subject matter areas, special transformation will be required. Such specialized transformation should be easily added to the authoring system, and even provided to the author automatically once it is known that they are used with high frequency.

The most versatile transformation of those noted above is the replacement of a defined string with another string. This may be required when one or more characters in the input are also the same as those which are used in specification of the answer standard. For example the input (A+B) may require the standard ((A+B)) in which the outer brackets are part of the format for specifying the standard. In this case the author may wish to replace (and) with / and \ to produce /A+B\ so that ambiguities are not encountered during parsing required in answer judging.

Words such as "a", and "the" are not likely to be important in judging an answer and may be removed at the option of the author. A standard system list could be available as well as a list defined by the author. The author should be able to determine whether words had been removed.

Special characters which negate each other include, for example, a space and a backspace, an index immediately followed by a reverse index. These may complicate the answer judging process and should be removed.

The transformations defined should also be made applicable to the standard set by the author as part of the answer judging process. The author's sequence of invoking the transformations is important and must be preserved.

Figure 13.8 illustrates the functional units of the transformations.

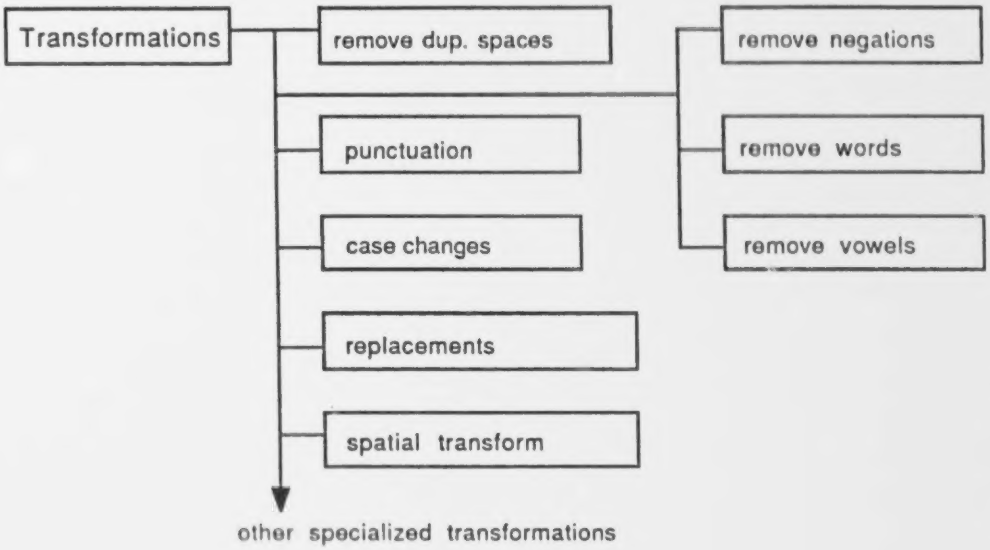

Transformations
remove dup. spaces
punctuation
case changes
replacements
spatial transform
remove negations
remove words
remove vowels
other specialized transformations

Figure 13.8 Transformation Functional Units

13.2.13 Answer Analysis Functional Units

The functional units required for non-trivial applications of CAI are likely to be very wide ranging in algorithmic complexity, subject matter application, and instructional objectives. Demands of authors could require algorithmic complexity approaching that required in language translation. Instructional objectives, particularly when the author wishes to accommodate a diagnostic analysis of a student's input, would mandate even more complexity. The kinds of answer judging required in grade three arithmetic is of a different quality than that required in genetics. The type of answer judging will also depend upon the nature of the student input, i.e., keyboard, pointing, or drawing. Although some general procedures can be defined, the acceptability of the authoring system in terms of answer judging is likely to be defined by the specialized functions which can be developed for an author.

The basic problem is that student input can be of almost infinite variety, and from this domain the author must be able to identify a specific input, and if necessary carry out a diagnosis of any faults which it represents.

The general procedures which the author would use for judging a student's answer are suggested to be the following:

1) Identify the nature of the input, e.g., string, value, pointing

2) Specify a standard for the mode of input

3) Apply any transformation to the input, or the input and the standard

4) Assess the closeness of the input to the standard

5) Decide whether further transformations and further assessment of the closeness to the standard is required

6) carry out any diagnostic assessment if appropriate

7) In light of the closeness of the input to the standard, decide upon an appropriate action, e.g., the nature of a reprise to the student or diagnose

The author must also decide how the student's response is to be categorized for purposes of recording the student's performance, or take advantage of authoring system default conditions. For example, in analyzing the responses of a set of students to assess the adequacy of the courseware, the author will want to know how the students' answers were categorized with respect to a particular question. The performance record will also be required to contain the original student input so that any ambiguities in the judging process can be identified and corrected as the courseware is optimized.

Factors which are important to analyzing string input from the keyboard are likely to be the following:

1) Was the student's input terminated by a "return" key, a time out, or because the length of the input was exceeded?

2) Was the input null?

3) Are word boundaries important? If so what will the delimiter be? For example, is a word to be defined as bounded by the space character?

4) Is the analysis to be made on words units, or character units?

If the string input is expected to be an equation, then further factors need to be considered:

1) Is the equation to be evaluated numerically?

2) What will be the word boundaries, arithmetic operators or spaces?

3) Over what range of values will the numerical evaluation take place, keeping in mind that some functions are not defined for some parameter values?

4) Cursor positioning becomes important, e.g., reverse index to indicate a superscript must not be removed by a transformation.

If the string input is to be treated as numerical values, then:

1) How will the author distinguish between the input of a single number from a vector?

2) Should the author evaluate each element of a vector separately and do this repeatedly for each element, or should analysis be done over the whole vector?

Because of the complexity which is involved in analyzing a student's response, the author should be able, during the authoring phase, to immediately test the approach taken by providing specific pseudo input for analysis. A graphical representation of the logic of the answer analysis process can also be helpful to the author particularly to show the path ways which various answer inputs might generate.

If input is a pointing response, then the author must be concerned with defining the tolerances of the boundary defining an acceptable point. In some instances more than one boundary will be required to accommodate different degrees of acceptability.

In the case of a drawing response, previous research (Lee, 1983) suggests one approach which would cover a wide range of simple drawing moves by the student. However, the nature of the answer checking in a geometry course is likely to be more complex than for a course in algebra which only requires the drawing of straight lines on a defined coordinate system. Further complications are introduced by such problems as the scaling of a drawing and its orientation on the screen. For example, if the student was asked to draw a right triangle, the triangle drawing can vary in size and in orientation on the screen. How should the author proceed to determine whether the answer is acceptable?

If there is any facet of authoring which suggested that one general authoring system **cannot** accommodate all authors, it is the problems associated with analyzing the student's input. The answer analysis problem does suggest however, that the authoring system must be constructed of highly modular software systems, components of which can be invoked for a particular class of authors, subject matter areas, and even instructional strategies.

Figures 13.9(a)-(c) illustrate some of the functional units required for answer analysis.

Provisions should be made to allow the author to set a number of answer analysis default parameters. These parameters are not necessarily used in every type of analysis which may be called for by the author. Some functional units which are useful are given in Figure 13.9(b) and are identified with the notation FUA in Figure 13.9(a).

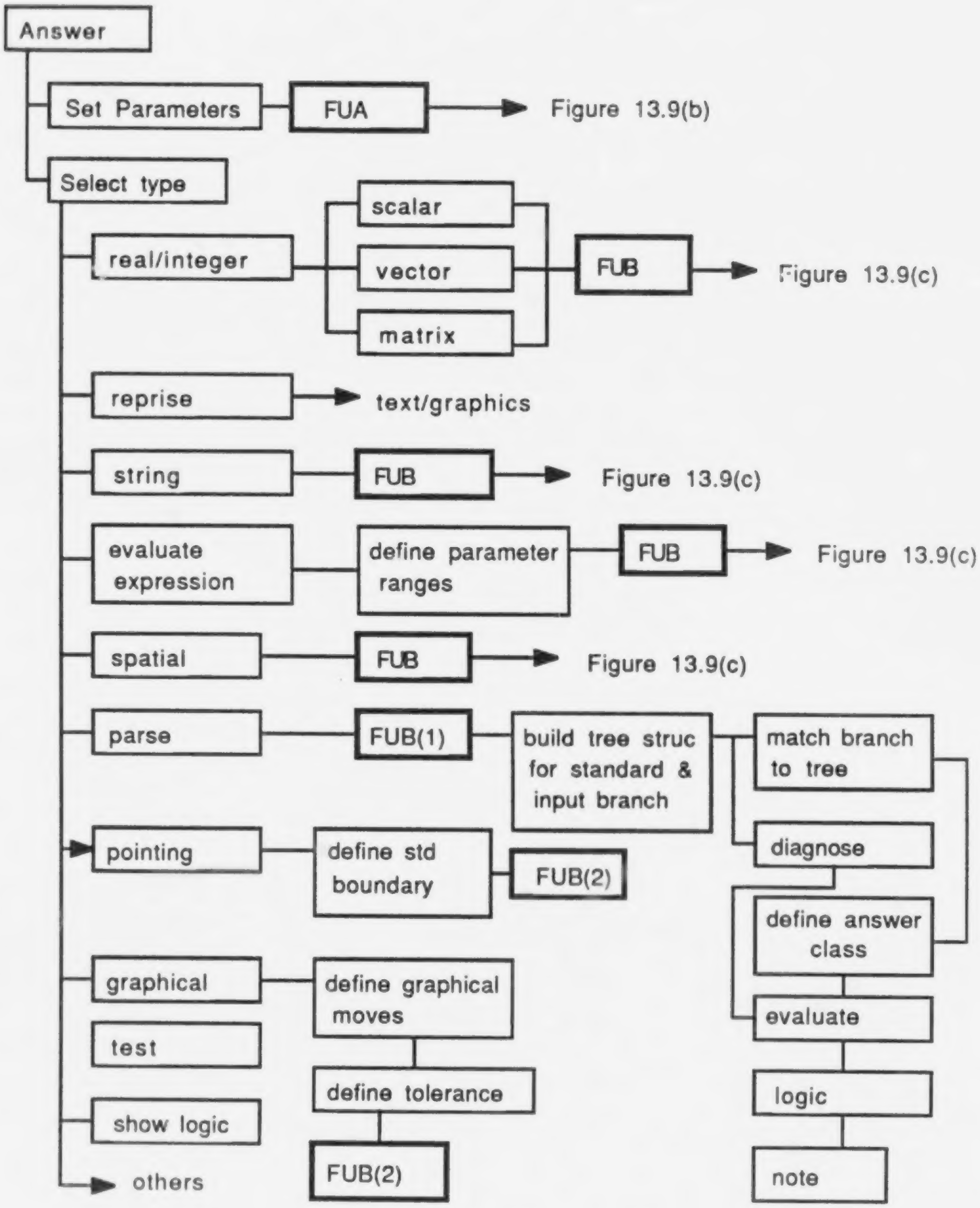

Answer
Set Parameters
FUA
Figure 13.9(b)
Select type
real/integer
scalar
vector
matrix
FUB
Figure 13.9(c)
reprise
text/graphics
string
FUB
Figure 13.9(c)
evaluate expression
define parameter ranges
FUB
Figure 13.9(c)
spatial
FUB
Figure 13.9(c)
parse
FUB(1)
build tree struc for standard & input branch
match branch to tree
diagnose
define answer class
evaluate
logic
note
pointing
define std boundary
FUB(2)
graphical
define graphical moves
define tolerance
FUB(2)
test
show logic
others

Figure 13.9(a) Functional Answer Units

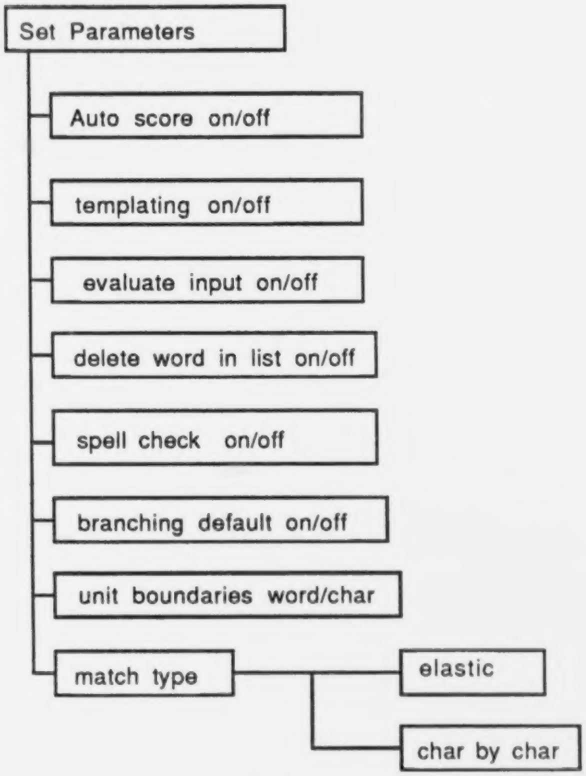

Set Parameters
Auto score on/off
templating on/off
evaluate input on/off
delete word in list on/off
spell check on/off
branching default on/off
unit boundaries word/char
match type
elastic
char by char

Figure 13.9(b) Sub Answer Unit FUA

With reference to Figure 13.9(b) the following brief description is provided regarding the functions listed:

1) Auto Score: This feature allows the author to have score keeping automatically enabled. Consideration has to be given as to whether the variables associated with score keeping are global or local to the instructional unit.

2) Templating: Since answer analysis can involve repetitive procedures, it is in this area in which the creation of a template is most useful. The template or pattern of moves required in the analyses is then made re-executable, with provisions for the author to specify unique information.

3) Evaluate Input: This feature enables the author's access to a report of such input parameters as the length of the input, whether the student's input was terminated with a "return" key, whether it was terminated because of the maximum string length being reached, or whether a time out terminated the input.

4) Delete word in list: When this feature is enabled those words in the author's list of words to be removed, are removed from the student's input. This feature should provide a report which the author can interrogate.

5) Spell check: A spelling checker is useful in some instructional environments. Consideration must be given to the nature of the algorithms used and to whether the computational cost is worthwhile (Nesbit,1985).

6) Branching default: The meaningfulness of this function depends to some extent upon the internal structure of the course code. This feature can transparently provide for a branch ahead when a correct answer is identified, and a branch back to repeat the input when a wrong answer is identified. Having this feature allows the author to concentrate on the units of analysis.

7) Unit boundaries: In algorithms which may require parsing, this feature defines whether units, such as words, are separated by spaces or some other special character. In the case, for example, of an answer which is an equation, unit boundaries might be defined as any of the four basic arithmetic operators.

8) Match type: Match type may be elastic or character by character. For example, in the "elastic" match the character unit "abc" would be reported as being found in the string "abdfc".

Figure 13.9(a) suggests that the main types of analysis which should be considered are as follows:

1) real/integer: numeric values of type integer or real, which may also be in a vector or matrix structure need to be identified.

2) reprise: this refers to the reply the author wishes to give to the student contingent upon the identification of an input. It should provide complete access to the creation of text and graphics on the screen, and should not be limited to a particular area of the screen.

3) string: this refers to textual or any alphanumeric input. This input might be a character, word, phrase, sentence, or even an equation; an equation might better be handled by a spatial analysis than by the procedures of the string analysis.

4) evaluate expression: an equation entered by the student, and a standard by the author, may be compared by assigning the same values to the unknowns in each and evaluating the two results.

5) pointing: when a pointing response is to be identified, the author need only specify the boundaries of the target area. Transformations are not meaningful in this instance.

6) graphical: the identification of the appropriateness of a graphical response may depend very much upon the capability of the terminal equipment used by the student. One approach (Lee, 1985) provides the student with choices of graphical moves, and these moves then become part of the analysis procedure, e.g., a circle could only be drawn by selecting the option "circle" and not any other option. Once it is known a circle was drawn, other parameter of the circle could be checked. Where linear functions are to represented in a two-dimensional coordinate system, the author can draw the appropriate function as a basis for defining an answer, and as well, draw the tolerances which would be acceptable.

Figure 13.9(c) illustrates the common application of the units required to define a standard and comparing it to the students input.

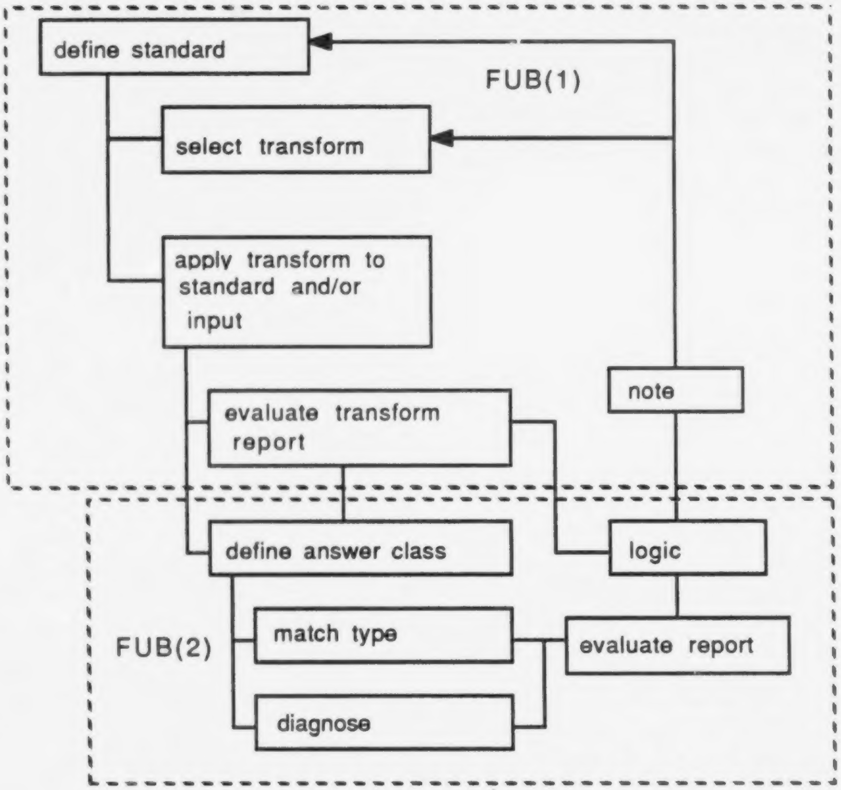

define standard
FUB(1)
select transform
apply transform to standard and/or input
note
evaluate transform report
define answer class
logic
FUB(2)
match type
evaluate report
diagnose

Figure 13.9(c) Sub Functional Units FUB

The following functions are performed by each unit:

1. define standard: the author must be able to enter a standard against which the input will be compared. Where generative algorithms are used, e.g., in an arithmetic drill, the standard may be computed by the algorithm and automatically provided at this point.

2. select a transform: the author selects a transformation considered appropriate, e.g., a spatial transform because an equation is to be identified

3. apply transform to standard and/or input: the transform can be applied to the student's input, to the author's standard, or to both

4. evaluate transform report: the results of the transform may be identified if meaningful in the context of the problem. For example, if no spatial rearrangement took place the equation contained no superscripts or subscripts. At this point the author may wish to take some action, or proceed to define a match condition.

5. define answer class: the author identifies the class to which the identified answer will belong, e.g., it might be simply "correct" or "wrong".

6) match type: this refers to the nature of the matching algorithm to be used and is followed by an evaluation of the matching algorithm's report.

7) diagnose: this refers to a detailed assessment of the results of the matching process which can be used to diagnose the nature of a student's error.

8) note: the author should be asked to provide a brief descriptive note of what the procedure has identified. This becomes important for purposes of documenting the instructional map of the course.

13.2.13.1 Logic Flow Chart

Because of the complexity of various forms of answer analysis which the author may wish to carry out, it is desirable that a flow chart representation of the logic created be made available to the author as an option. This flow chart should be annotated by the notes which were requested of the author during the process of defining the procedures. Ideally, the author should be able to call upon the degree of "granularity" he wishes to see, starting with a representation of the logic at the most general level, and an examination of each step in the processes at the most specific level.

13.2.14 Arithmetic Operations

Arithmetic operations are frequently required during the execution of the courseware, and therefore must be specified by the author. In addition, it is useful for the author to have access to the same arithmetic operations during the authoring process. Some arithmetic operations for control of the cursor position on the screen may be required, as for example, when the screen position of an object has to be calculated. Arithmetic operations are also required by the evaluate procedure during answer analysis, and for some types of decision making.

How much arithmetic power should be made available to the author is of course, a function of how much power the courseware interpreter has and the nature of the subject matter content being defined. When a considerable amount of power is required, consideration needs to be given to creating this power as a direct part of the authoring system or drawing upon an existing software system such as Fortran or APL. Of course, the same argument can be made for any sub-system of the authoring system.

Although scalar arithmetic is a minimum requirement, consideration should be given to vector and matrix operations. In addition, such functions as square root, random number generation, trigonometric functions are required.

In interfacing the author to arithmetic operations, the following should be considered:

1) Variable types: What types, e.g., integer, real, binary, are to be provided? Is scalar and vector type operations to be available?

2) Types of Operations: Although the basic arithmetic operations of addition, subtraction, division, and multiplication are straight forward, their application to vector and matrices may require special definition.

3) Syntax: Where more than trivial operations are to be made available, then the syntax and precedence order of operations must be defined. When the syntax is improper error messages are required for the author.

4) Functions: What functions should be provided? In most instances random numbers, square root, logarithms, and trigonometric functions are required. These functions can fail under specific circumstances and consideration must be given to default conditions during run-time, and the saving of error messages for later analysis.

5) Author Naming of Variables: If the author is allowed to provide names for variables created, than consideration must be given to keeping a list of these names, their types, and associated author notes. Duplicate names cannot be allowed.

6) Errors: As noted in item (4) above, errors in computation can occur during the authoring phase as well as during the running of the courseware. Consideration must be given to what default conditions will be invoked during run-time. Ideally, the student should never encounter a run-time arithmetic error, but until all bugs are removed special actions need to be taken.

7) The arithmetic operations would be very useful in conjunction with plotting software, e.g., an author defines a function to be plotted by its equation and the range of the values of the parameters making up the function.

The basic units of providing the author with arithmetic capabilities are given in Figure 13.10.

With reference to Figure 13.10, the units clear, save, and rename require further explanation. The function clear is suggested to allow the author to clear a set of variables rather than to clear each individually. For example, the author should be able to set to zero variables identified as a list, or by indices. The save function is suggested because to define a complex arithmetic computation is time consuming and prone to error. The save function would allow the author to save the definition by name for later recall and use. The rename function is provided to allow the author to rename a variable or the name of an arithmetic definition.

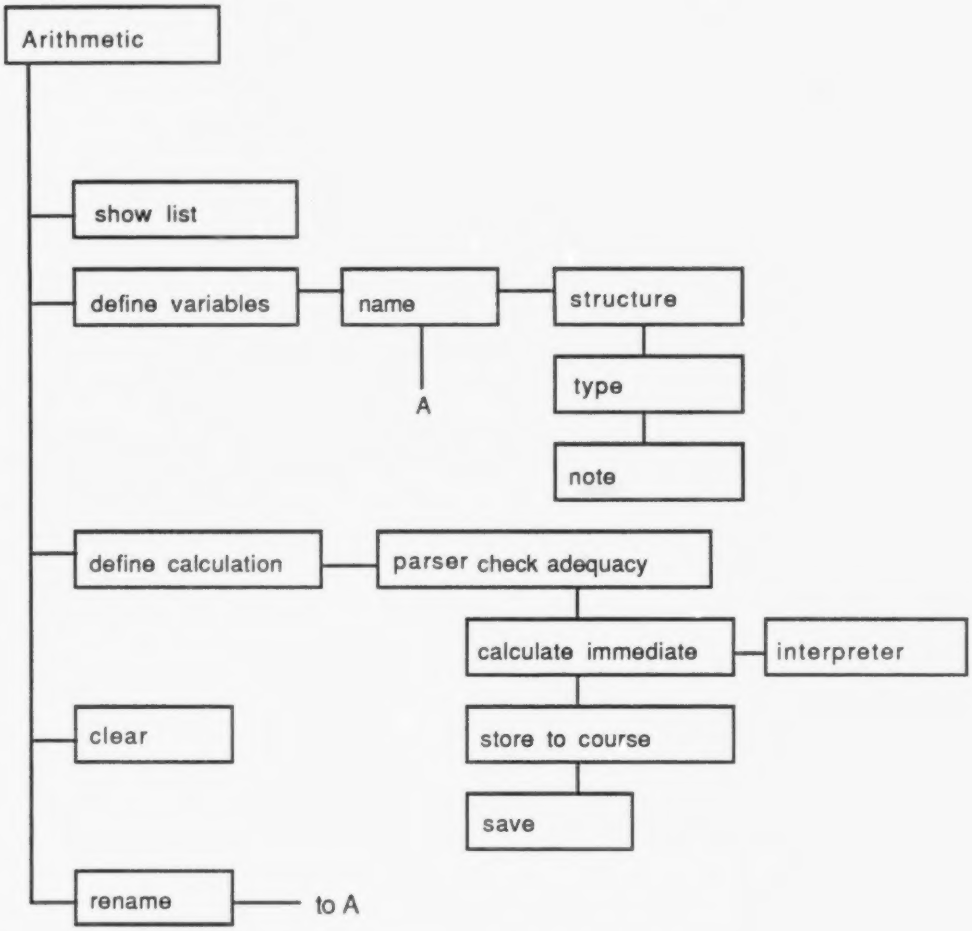

Arithmetic
show list
define variables
name
structure
type
note
A
define calculation
parser check adequacy
calculate immediate
interpreter
clear
store to course
save
rename
to A

Figure 13.10 Functional Arithmetic Units

13.2.15 Templating

Templating refers to the capability of the author to create an instructional sequence (a pattern), and to save it for repeated use. This implies, of course, that certain components of the actions templated are undefined within the template, but are later defined by the author when the template is executed. For example, during the definition of a template, the invocation of the process to enter text to the screen would be saved as would the cursor position at which text was entered, but the text entered would not become part of the template.

Some authoring system attempt to provide this feature by a "fill in the form" mode, but the forms are generally quite restrictive, are pre-designed and rigid, and cannot be modified by the author.

In order to not restrict the nature of the template which can be created, it should be possible to save any sequence of commands used by the author as a template. It should be possible to construct a template for a micro instructional sequence such as would be used within an answer analysis sequence, or over a large procedure such as in the creation of an examination, by having the author define the sequence once.

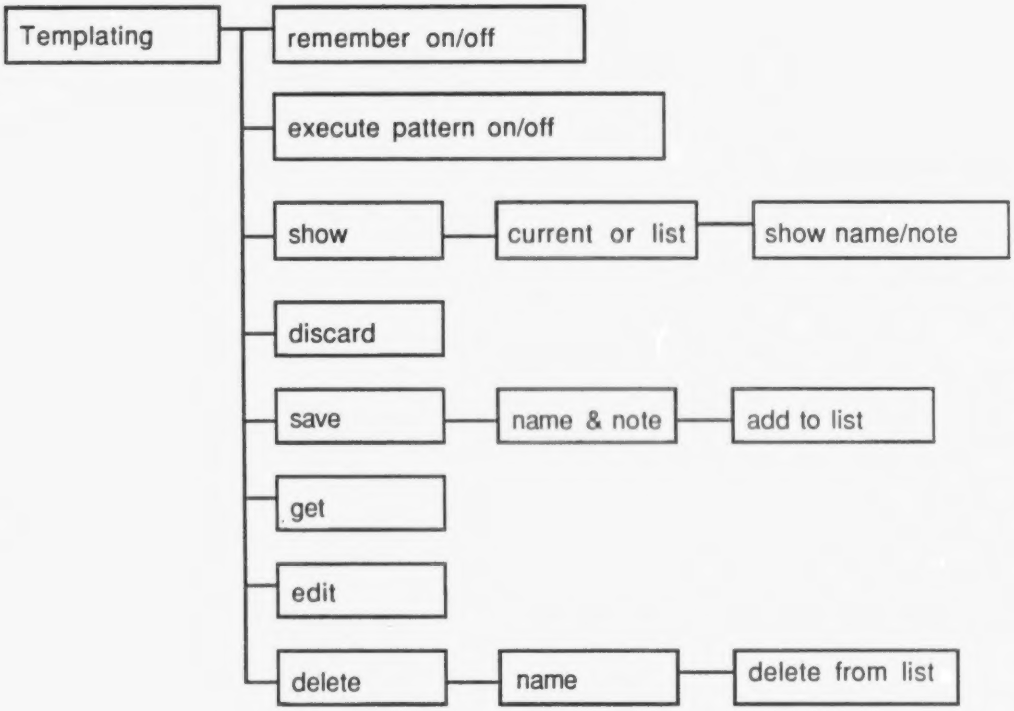


Figure 13.11 Templating Functional Units

The major functional units for the templating process are listed in the first level of boxes in Figure 13.11. The "remember" option starts the process of saving the author's actions, while "execute" invokes the template to recreate the instructional sequence. Since it does not make sense to have a pattern of the whole course being created, a visual indication should always be available for the author signaling that the templating option is enabled. The "show" option allows the author to see the current list of steps in the current template, or a list of the names of templates which have been previously created and "saved".

The "discard" option allows the current template to be discarded. To prevent errors, a current template should be explicitly discarded before a saved template is recovered. It should be possible to save a current template under an author defined name. Duplicate names would not be allowed. The author should provide an explanatory note when a pattern is saved. "Get" retrieves a named template.

"Edit" allows the author to edit the steps in a current template. This is suggested because frequently a small modification can be made to a template, such as changing the position of a cursor or changing a "correct" category to a "wrong" category, and the template can be used for another purpose. The usual edit actions such as replace, delete, and insert are required.

The last delete option in Figure 13.11 is suggested in order that the author can delete a named template from the stored list of templates.

Of course, all the actions noted should be possible without permanently destroying the major portion of the screen display which the author is authoring.

13.2.15 Comment

It is highly desirable to provide the author with a comment feature which allows attaching to that point in the course code an explanatory note of the author's choice. Such comments may relate to course content, logic being used, or locations in the courseware where special attention should be directed while testing.

13.2.16 Run and Edit

The capabilities and flexibility of these two functions depend very much upon the internal structures used to store the course representation as it is being created by the author. Of major concern is the definition of the domain of the course over which an adequate execution and edit can be carried out, especially if the author is to be allowed to move between course execution and editing within the authoring system environment.

Ideally, the author should be able to execute the course and interrupt the execution in order to make changes, observe the result of the edit, and to continue execution of the course. Editing may be required on those components of the course which are directly observable as effects on the screen, or they may be required on those components which are not directly observable on the screen, e.g., a change to the transformation applied to the student's input. An adequate execution may depend upon the definition of some variables, which are defined outside of the span of execution.

An important feature for an experimental authoring system having more than trivial complexity is that there should be more than one method of editing the internal representation of the course. One entry to editing would be through the software of the authoring system, another might be a regular editor accessed through the operating system commands of the computer being used. However, the latter editors are not likely to work when special data structures, rather than line files are used to store the software created by the author. Providing the author with only one editor, is like providing a house with only one door, which unfortunately if it becomes locked, access to the house becomes impossible. Such restricted access is more typical of micro computer systems.

13.2.16.1 Authoring and Editing

Regardless of how courseware is stored internally, the authoring system should be designed to minimize errors. Appropriate error sensing, trapping and messaging is important, but can be complex when the author is deeply embedded in a tree structure. Solutions to error messaging in such cases have been proposed by Efe (Efe, 1987).

Errors in authoring actions resulting in screen representations are easily identified, but the difficulty of their corrections depends upon the need to access special data structures used to store a representation of the courseware. It is easy to extrapolate from the visual editing systems used on micro computers with word processing software to desirable features for an editor for courseware development, but it must be recognized that generally there exists a one to one relationship between what is observed on the screen and internal representations of text in word processing editors. Furthermore, the latter represent strictly linear sequences of text.

The major concerns which must be recognized in developing one or more editors for use as part of an authoring system are as follows:

1) How is the internal representation of the courseware to be stored? Is it in a linear sequenced line file or in some specialized data structure such as a tree structure?

2) What text editing capabilities should the editor have:

    a) Should the editing be done directly on the item to be changed or some transformation of the item?
    b) Should the editor provide such features as search for string, search and replace a string, cut and paste?
    c) How should text attribute changes be made, e.g., color, and font changes, and over what domain of the text should the change be invoked?
    d) Is repositioning of blocks of text to be provided?

3) What graphics editing capabilities should the editor have? Questions similar to those posed in (2) above are relevant.

Correction of observable elements displayed on the screen but which were created several authoring steps earlier will require a pointer to their internal representation. The problem of identifying the source representation can become more complex when the screen holds many independently written elements and their positions defined from an absolute location through the use of off-sets. The location of the current absolute location and its off-set is required. If the next writing position to the screen is dependent upon the last item, removal of the last item could cause havoc to the positioning of all subsequent elements. Such problems as described above can be made more tractable if separate windows are used and each window does not hold cumulative elements for display, i.e., when an element is accumulated to a window a new window is written to the screen.

Non-observable elements created during the authoring phase present an additional level of complexity. Non-observable elements would include such things as arithmetic or logical operations. These elements must be identified as existing for the author, and be presented to the author in a meaningful manner, i.e., transformed before representing them to the author. One option is to allow the author to break from the normal authoring actions and view an iconic representation of the course in which the unobservable elements have their special icons. An example is given in Figure 13.12.

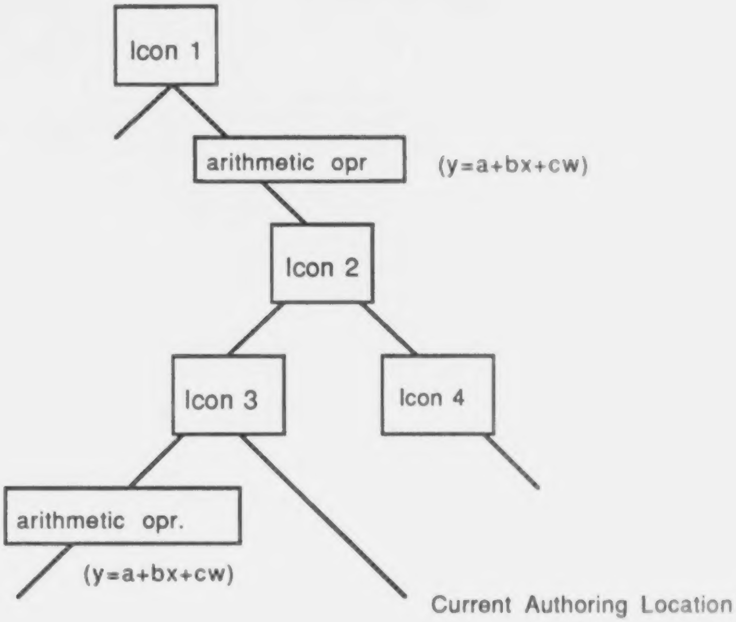


Figure 13.12 Example of Iconic Representation for Unobservable Component in Courseware

Where many different classes of actions are used by the author, the number of such icons can begin to mushroom resulting in the author being unable to discriminate amongst them. In this situation the most meaningful "icons" are likely to be words! The iconic representation although useful for representing small domains may be inadequate when a lengthy course having literally thousands of iconic units has been constructed. This problem suggests that hierarchical levels of representation should be used.

The editing of graphic elements presents the same problem -perhaps more so as the granularity of the coordinates is much finer than that used to define position of text on the screen. In addition, relocatable graphics depend heavily upon off-set coordinates which add further complexity unless the initial elements are constructed with absolute coordinates and later converted to relative coordinates.

One approach to the editing problem is to allow the author to step back over the authoring actions in the sequence in which they were created. This is best done, however, by executing the course and then interrupting the execution to enter an editing phase.

13.2.16.2 Executing and Editing

Executing the courseware and then breaking to enter an edit mode makes some problems more tractable because the precise location of the last observable element displayed is more easily known. However, execution requires the definition of a starting point in the courseware and that point may be beyond the definition of some variables needed for an adequate execution. The author would then be required to provide a default of some kind. Two disadvantages of this approach include that it may take some time for the author to reach the point where the editor should be invoked, and some representation has to be given to those elements which do not produce a screen effect. These problems suggest that special consideration has to be given to the environment in which the execution of the course takes place. Features which would likely be required include those required in a debug mode, but might also include such features as assigning a specific condition or a default condition at decision or branching points. For example, the author might request an execution of the course with all branching based on a "correct" answer being given. This is also not without problems because frequently authors have been known to use the branching default logic of a "correct" answer when the reprise text to the student is not for a correct answer! Notwithstanding such problems, such a default condition as described would help increase author's efficiency.

13.3 Need for Research

The complexities of developing an editing system within the context of an authoring system are not minor. Some tentative solutions are evident in existing authoring systems but much more work of trying out alternatives is required. The complexities noted above are compounded when the use of video disk and audio sound generation are considered as candidates for modification through editing procedures. It is possible to have a single editor to handle text and graphics, but this would require the context of the edit action to be known which implies that the editor "knows" of the form of the data structures within which the changes are to be made.

Chapter 14
Implementation

14.0 Introduction

Two implementations where made of an authoring system. The first implementation described in this Chapter was developed to explore the author-machine interface, while the second was developed to explore the specialized data structures required to support an internal course representation superior to that of the first implementation,and related authoring functions. Appendix A contains photographs of screen displays which are presented to the author, while Appendix B is the author's manual for use of the authoring system.

The first implementation was designed to be integrated with an existing CAI system which has been in operation at the Division of Educational Research Services for a number of years. The capability of developing the authoring system in the context of an existing CAI system had a number of advantages:

1) The authoring system could draw upon components of an existing system which were already programmed, in particular a stable interpreter for a course. (The advantages of this approach have been noted by Kaiser and Garlan, 1987).

2) The context of the authoring system was such that although experimental, it had to create courseware which would operate in a practical environment.

3) The authoring system had to co-exist without faults within an existing CAI system.

4) The author would be required at various stages in the development of courseware to act as a student as well as an instructor for purposes of complete testing. The systems for the latter two types of users already existed.

There are of course disadvantages to this approach. The major disadvantage is that some constraints are placed upon the design of the authoring system which would otherwise not be present. These constraints in some instances accentuate the problems of the author interface which otherwise would not have arisen.

The existing CAI system used an internal representation of a course which was a modified form of the Coursewriter programming language. The internal representation of the course was highly linear. Branching was controlled by explicit statements or implicitly through the use of an internal logic table. The interpreter for this representation has been highly stable and is used for the delivery of university credit courses. The interpreter is a small part of the overall CAI system.

Figure 14.1 illustrates the position of the authoring system in the context of the student and instructor support systems.

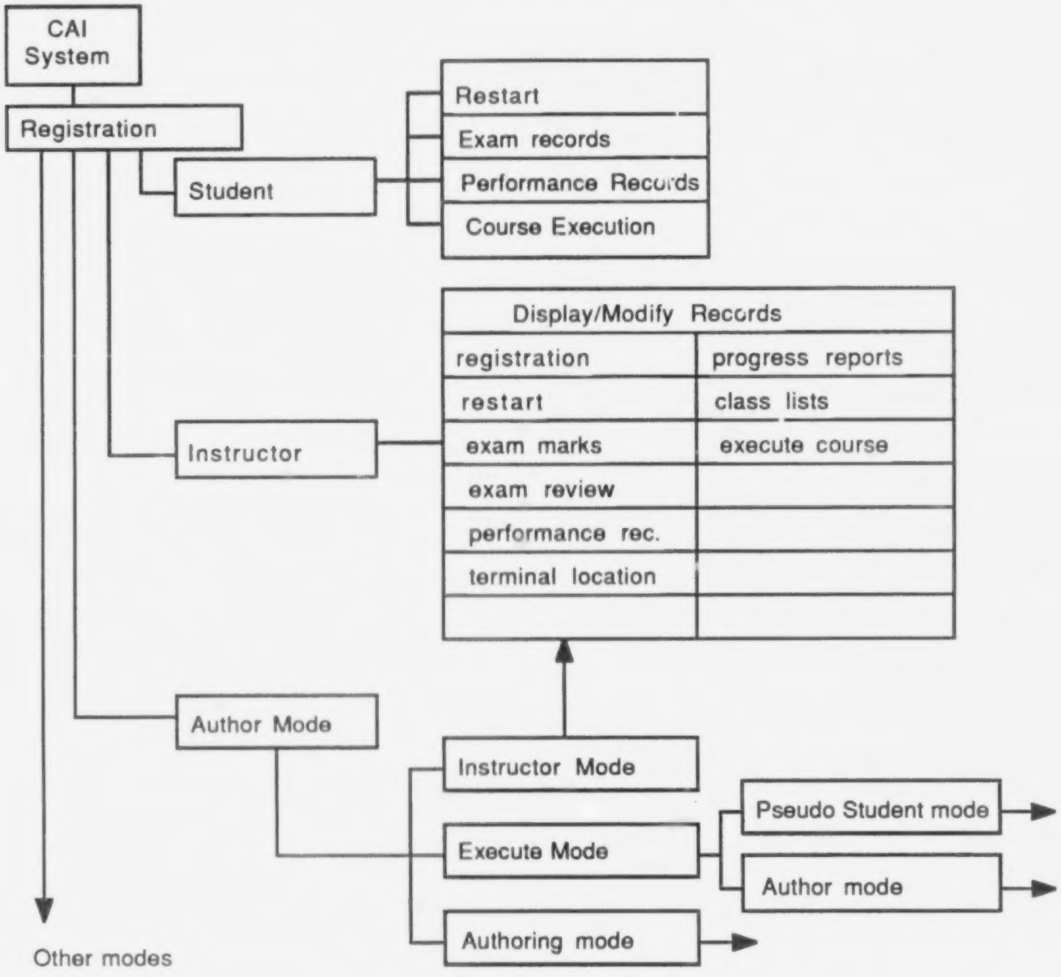

CAI System
Registration
Student
Restart
Exam records
Performance Records
Course Execution
Display/Modify Records
registration
progress reports
restart
class lists
exam marks
execute course
exam review
performance rec.
terminal location
Instructor
Author Mode
Instructor Mode
Pseudo Student mode
Execute Mode
Author mode
Authoring mode
Other modes

Figure 14.1 Authoring System as Part of CAI System

With reference to Figure 14.1, based upon registration records the CAI system software identifies the nature of the user as a student, instructor or author. Since the author will require to take the role of the instructor and the student for purposes of testing the courseware generated, access is permitted to these two modes. Although courseware execution is provided from within the authoring system, a direct access to execution is also provided in order to increase efficiency.

14.1 Entry to Authoring: Confirming Information

When non-trivial courseware is being created, an author seldom works on a single section of the courseware. In addition, an author frequently wishes to try out an idea without making a commitment to using the courseware. For such reasons, once the author has defined a section of the course to be authored, information about the current status of the section to be authored should be presented. Some of this information is obtained from files which provide continuity from one authoring session to another. In addition some parameters of the authoring system should be open to modification by authors having a sufficiently high privilege level.

Figure 14.2 illustrates the type of information which is presented to the author upon entering a specific section of a course.

With reference to Figure 14.2, the current implementation of the authoring system provided the following sequence of events when the author first entered the authoring phase:

1) An author user file was read which contained in addition to other parameters, the privilege level assigned to the author. This was a parameter in the range of 0 to 9, with the higher number providing the greatest range of privileges.

2) The author was then asked to define the number of the course section or chapter to be authored. This information together whether the author's registration which provided the course name, allowed the definition of all the file names needed during the authoring process. The author is not aware of the various files and their names.

3) The author user file contained four other parameters in addition to the privilege level. These parameters and their options were as follows:

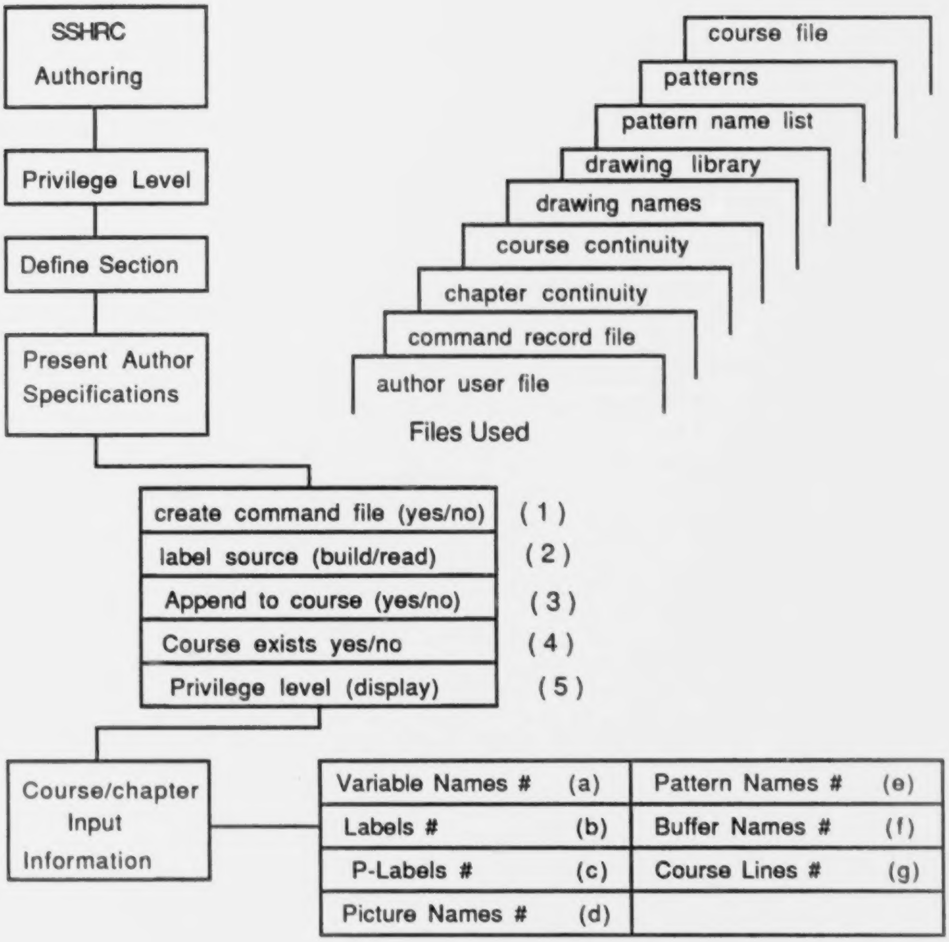
   a) command: yes, no, or query
   b) labels: build, read, or query
   c) append: yes, no, or query
   d) comment:

   The words following the colon indicate those attributes which each parameter could take. For example, "command:" could be followed by yes, no, or query. If the privilege level was 9, and if the attribute was query, the author was asked to provide the attribute yes or no for the current authoring session. In all cases, if the attribute was query, and if the author privilege was sufficiently high, the author could set them himself. If the author's privilege level was low, these parameters are fixed at registration and cannot be changed by the author.

   The attributes following "labels:" allowed the author to either "build" a new table of labels by having the current course searched for all labels, or to "read" the current labels from a file.

   The author with sufficient privilege could either append the current material to be authored to the existing material, or create a separate section which could later be merged with an existing section.

   In the case of the lowest level of user, which in our testing were undergraduate students learning about courseware authoring, "create:" was set to yes, "label:" to build, and "append:" to yes. The "command:" parameter was set to yes in order to be able to track all the moves the student authors made during the authoring process. This proved to be an invaluable aid in confirming and clarifying inappropriate moves made by the students. Parenthetically, it should be noted that seldom can an author, even an experienced one, precisely describe the actions which were taken even five minutes previously.


SSHRC
Authoring
Privilege Level
Define Section
Present Author Specifications
course file
patterns
pattern name list
drawing library
drawing names
course continuity
chapter continuity
command record file
author user file
Files Used
create command file (yes/no) (1)
label source (build/read) (2)
Append to course (yes/no) (3)
Course exists yes/no (4)
Privilege level (display) (5)
Course/chapter Input Information
Variable Names # (a)
Labels # (b)
P-Labels # (c)
Picture Names # (d)
Pattern Names # (e)
Buffer Names # (f)
Course Lines # (g)

Author Specs	Parameter	Input	Entries
Privilege Level	9	Variable Names	5
Command Record	Query	Labels	8
Label Source	Read	P-Labels	0
Append Course	Query	Picture Names	50
Course Exists	Yes	Pattern Names	2
		Buffer Names	3
		Course Lines	75

Figure 14.2 Initial Information Presented to Author

In the case a chapter had not yet been authored the label: and append: parameters were appropriately defaulted. In addition, the logic of the parameters as a set were checked regardless of the author's privilege level. For example, if a chapter did not already exist, the labels could not be defined by searching through the internal representation of the chapter.

The author was informed automatically whether a chapter of the course he was entering already existed or not.

4) Summary information was then presented to the author. The summary information contained the number of variables which had already been defined, the number of labels defined, the number of labels which had yet to be defined, the number of pictures and patterns (templates) which had been created in the course regardless of the section in which they had been created, the number of storage buffers which had been defined, and the number of lines in the section of the course which was to be authored. The latter item, the number of lines provided the most useful information to the author confirming that the appropriate section of the course had been entered.

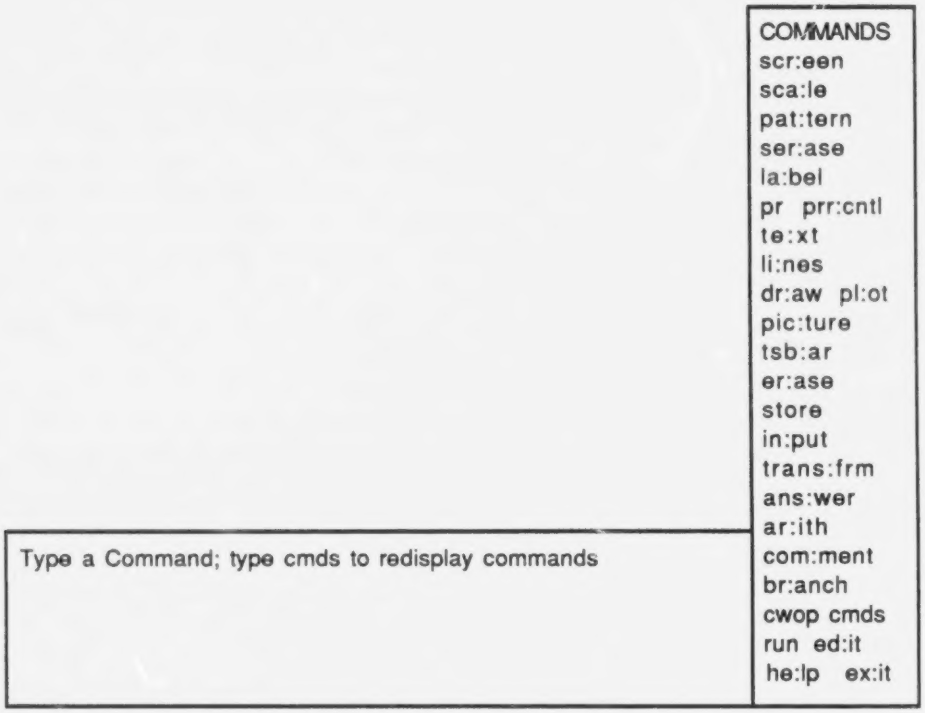
The files which are automatically created by the authoring system are also included in Figure 14.2. The author was unaware of these files. The *chapter continuity file* contained information pertinent to the authoring of a specific chapter. One such file existed for each chapter. The *course continuity file* contained information pertinent to the whole course, for example, the list of names of drawings created were in this file since a drawing was available for display in any chapter of the courseware being created. The same was true of patterns or templates created.

It is important to note that a fair amount of information was being stored for the author to provide support that the intended chapter of the course was being entered, and to support the author during the authoring process. For example, the names of drawings cannot be counted on as being accurately kept by the author, thus the system kept not only the names of the drawings created but also an associated descriptive comment. The same was true for variable names, labels, buffer storage, and pattern names. All these names and their associated statements could be recalled during the authoring process so that the memory load on the author would be reduced.

Figure 14.2 also provides an example of the screen display which is presented to the author showing the specifications the author may reset (Query) and summary information about the course chapter being entered.

14.2 The Author's Main Command Menu

Upon entry to the main section of the authoring system the author is presented with the display shown in Figure 14.3.



Type a Command; type cmds to redisplay commands

COMMANDS
scr:een
sca:le
pat:tern
ser:ase
la:bel
pr prr:cntl
te:xt
li:nes
dr:aw pl:ot
pic:ture
tsb:ar
er:ase
store
in:put
trans:frm
ans:wer
ar:ith
com:ment
br:anch
cwop cmds
run ed:it
he:lp ex:it

Figure 14.3 Main Authoring System Menu

The menu of commands which appears on the right hand side of the display are considered to be the major moves required to create instructional sequences. These may be grouped into the following major categories:

a) control of the screen
b) the creation of patterns or templates of instructional sequences
c) the control of branching and student restart point
d) the creation of screen displays using text and lines
e) the creation and display of graphics and plots
f) student control of screen display presentation
g) the storage of strings
h) the definition of student input
i) the transformation of student input
j) answer judging
k) arithmetic operations
l) executing and editing the course

With reference to Figure 14.3, those characters from the left to the colon (:) form the short form of the command. There are approximately 21 commands which are displayed in the menu. Although consideration was given to represent these commands as icons as is currently popular with some micro computer authoring systems, it would be difficult to create 21 icons with the degree of discriminability present in the words themselves. In this instance, the use of icons would place a unnecessary memory burden upon the author.

14.2.1 Functions of The Main Commands

scr:een

This command allowed the author to set one of three screen formats. The default setting was a 45 column screen, while the other two formats were a split screen and an 80 column screen. In all cases a maximum of 16 lines were made available, each line being defined as containing 2 rows. The latter feature allowed for true subscripting and superscripting. The split screen provided the author with a dynamic display area on the left half of the screen, and a static display area on the right side of the screen for display of fixed position drawings or plots. The same style of character font was available in all screen formats, only a change in the size of the characters was necessary to achieve this feature.

Note: The 45 column screen provided a easily readable font size, however, this size of font created problems when line by line algebraic derivations were required. Parts of the derivation would have to be extended beyond a single line in the 45 column format breaking the ease with which the interrelationships between subsequent steps could be observed. Although the 80 column format used smaller size fonts, it was considered more important to maintain the integrity of each step from a conceptual and learning point of view.

sca:le

This option provides a set of vertical scales in 5 character-width units and the line numbers 0,2, ... 32 to help guide the author in positioning text and drawings on the screen.

Note: Guidelines were deemed necessary because authors found it difficult to align text particularly near the bottom of the screen. (The latter is particularly true when the screen exhibits non-linear distortions). In addition, under the text mode the author in this implementation was required to set the initial row and column parameters for positioning of the cursor.

pat:tern

This option allows the author to save a sequence of commands which define an instructional sequence. In effect the author can create his own template by defining the sequence of moves once, having it stored, and then recalling it to repeat the sequence as many times as necessary. Where the author has to enter information which is variable, like text to the screen, the automatic execution of the sequence of stored commands is suspended and the author then inputs the required text. The sequence of commands forming the pattern may be saved under a name given by the author, retrieved and reused or edited.

Note: The pattern option provides the major advantage of allowing the instructor to create templates which can be used repeatedly. For some instructional sequences, such as in answer judging which tends to be algorithmic in the processes called, the use of templates increases efficiency and reduces errors.

ser:ase

There are occasions when the author wishes to erase the guidelines provided on the screen, as for example, when the author switches from a 45 column screen format to an 80 column screen for the lower half of the screen. This options provides for the erase of the last used guide lines.

la:bel

The internal representation of the courseware code required the use of labels. Labels are alphanumeric identifiers to which branching could be explicitly made, and also form the basis for arbitrary subdivisions of the courseware. An arbitrary subdivision of the courseware within chapters may be based on the subject matter being presented. This option allowed the user to either create a label explicitly or to have the system create the label. The author was asked to supply a note to be associated with the label.

Note: Although the author is allowed to create labels such as prob1, prob2, etc., this was found to be a disadvantage because after a certain point, the author could not create labels which were sufficiently different to be meaningful. Software provisions had to be made to prevent duplicate labels being created by the author. Since a chapter may have hundreds of labels, provision had to be made to allow the author to access a list of labels and their associated notes. Labels automatically generated were numeric consisting of the chapter number and three additional digits starting at zero and incremented by four. Although automatic branching was a feature of the authoring system implemented, the author could still define an explicit branch to a label which had not yet been defined. This required maintaining a list of "branch to labels" which had yet to be defined.

pr and prr:cntl

These two commands provided for control of implicit branching. For the major part of authoring a course only the prr had to be explicitly set by the author. The prr formed the point at which the student would be restarted at the next instructional session, and therefore could only be defined by the author in the context of the content currently being authored.

Note: Some authoring systems available on micro computers do not have the capability for defining the restart point for a student. One can only assume that such authoring systems are designed for the creation of short almost trivial sequences of instruction. The restart capability must be provided for any courseware of useful length.

te:xt

The text option provided the author entry to the creation of screen displays including the definition of lines which could be used for purposes of emphasis. The creation is made directly to the screen as it would be seen by the student. Since a major portion of the author's work in some courseware is the creation of text on the screen, this topic is considered more fully under Section 14.5.6.

Note: The capabilities of placing text to the screen as it would be seen by the student can depend upon the characteristics of the terminal used by the author. Consideration has to be given to cursor positioning, font changes, color changes, erasures, subscripting and superscripting. A major consideration is the extent to which editing can be done on any text during the creation phase. This relates to the problem of edit span discussed earlier.

li:nes

This facility was provided because the use of lines, circles, curves, boxes, etc., are frequently used by instructor for purposes of creating simple diagrams and emphasizing text, e.g., placing a set of words in a box, pointing from one location of the screen to another location.

Note: Consideration has to be given to the capabilities of the authoring and student terminal in the mixing of graphics and text in a common window or display area. In the case of simple lines, the definition of the attributes must be considered. A major problem is however, the editing of the lines created.

dr:aw

This facility allows the author to create drawings directly on the screen and in the context of text already on the screen. This facility would also be dependent on the authoring and student terminal capabilities. Drawings which could be located anywhere on the screen and those for use with the split screen can be created. Both types of drawings are available for re-use anywhere in the course. A single drawing may be composed of many smaller drawings.

Note: Because lengthy courses can have hundreds of drawings, a list of drawings already created with an accompanying note was required. This list needs to be available to the author in order that the appropriate selection can be made.

pl:ot

A plotting feature was provided for the author which would allow the construction directly on the screen of bar graphs, continuous line graphs, point plots, and vector plots. Provision was made to allow the author to construct these plots from simple data which the author could generate quickly (on the "fly") as well as from more complex data, e.g., data points for a normal curve. These plots were considered to be instructional plots and not the type of plots generated by plotting packages used for purposes of publication. Thus, considerable flexibility was given the author to create axes, to label or not label axes, to decide upon the value of each unit represented, etc. Up to five or six plots are sometimes required to be referenced as a single plot.

Note: The nature of the characteristics used in plotting data can be the subject of debate. The position taken in creating this facility was that plots as used by an instructor in the classroom are not the same as plots used for purposes of publication in a journal or the popular press. Thus, the important difference rests with the clarity with which the plot conveys to the student the concept to be illustrated. Our preference is for simple clear drawings without the "pizzaz" used in video games and in the creation of TV titles and advertising.

pic:ture

The picture facility provides the capability for the author to display on the screen drawings which have already been created.

Note: Access is needed by the author to the list of drawing names or ids, and their associated notes.

tsb:ar

This option allowed the author to provide for a pause in the presentation of the course during its execution until the spacebar or return key is touched by the student. It provides for student control over the rate at which the courseware is presented.

er:ase

The system which was implemented was not restricted to the display of complete and independent frames of information. A screen could accumulate as much information as the author wished. The erase feature allows the author to erase identified rectangular areas of the screen, to erase from one defined line to another, or the whole screen.

Note: The implication of erasing the whole screen, or over the complete width of the screen is that menus may have to be reconstructed for the author. In addition, since the number of columns the screen has is a function of the screen format set under the option "scr:een" noted earlier, the software must always know regardless of what process is being carried out, the nature of the current screen format.

store

There are many instances when an author wishes to store a string of text or alphanumeric characters, e.g., the storage of keywords to be used in answer analysis. In order to increase the efficiency of the authoring processes provision was made for the author to store a bank of such strings, assign each item stored a name and an associated note. In addition, the author is required to cause the storage of strings such as keywords to be placed into the courseware representation and thus, the authoring system must keep track of the storage allocations made within the course.

Note: The author may store an unlimited number of such strings, and thus provision must be made to allow the author to review the list and select the string currently required. Duplicate names cannot be allowed. Lists and the contents of storage locations placed into the course representation must be maintained over authoring sessions.

In:put

The input option allowed the author to define the location and input parameters of a keyboard response, or a pointing response by the student. Parameters such as number of characters to be allowed in the input and the amount of time to wait for the input are usually required. In the case of the keyboard input the author is required to define the area of the screen in which the input was to be received. In this implementation this was done by defining a rectangular area using cross-hairs.

Note: This implementation was not restricted to the use of single digit input such as required for multiple choice responses nor was it limited to a defined area of the screen. However, one must keep in mind that the longer the input allowed the student, the greater the problem of determining the meaning of that input. In this implementation font changes, superscripts and subscripts as input was not part of the course interpreter and so was not implemented in the authoring system.

Trans:frm

Input from a student can be of almost unlimited variety regardless of the question posed. Therefore, some cleaning up of the input, i.e., a transformation of the input, is required. The transformations may be quite simple such as the removal of leading and ending spaces, or a sequence of spaces, punctuation, and shift in case. In this implementation, provision was made for case shift, removal of extra spaces, and punctuation as well as replacing a substring by an author defined substring.

Note: The application of transforms is important since they allow for cleaning up the input prior to the application of more formal procedures of analysis involved in answer judging and diagnostic assessments. In some instances the transformations are important to the processes which follow. For example, it may be desirable to transform a reverse index, a character, and an index to another character identifying a superscript.

ans:wer

The answer option provides the author with the processes which are used to identify the student's input. In this implementation two major processes were implemented: a) Boolean representations of the answer being sought, and b) an arithmetic range check in which the author could identify which rank ordered position in a sequence of numbers was to be checked. An automatic score keeping facility for correct, wrong, and unrecognized answers was provided under option for the author. Implicit branching was provided for the author as a default, however, provision was made to allow the author to define an explicit branch. This was done by questioning the author as to where the branch was to go, and under what conditions, i.e., conditional and unconditional branching was allowed as an option.

Note: Checking the student's response can be a complex process, and to some extent is subject matter dependent. The author must have access to stored keywords to be used as the standard against which the student's input is to be judged, and access to arithmetic, branching, and the display of text and graphics. The display of text and graphics is required as part of a reply the author may wish to construct. The process of answer judging is sufficiently complex that visual supports are required for the author.

ar:ith

The arithmetic option allowed the author to create variables by name, and to carry out simple arithmetic operations. A list of the variable names and their associate notes is required to assist the author in recalling and using the variables appropriately.

Note: The arithmetic capabilities of the run-time system and the internal course representation govern to a large extent the nature of the arithmetic operations involved.

com:ment

A feature which is extremely useful to the author who has more than one method of accessing the internal representation of the courseware created, is the capability to store comments in the context of that code. This feature allows the author to use the search feature of an editor to quickly get to a desired location in the code.

br:anch

Although implicit control of branching was provided for the author, the option to define an explicit branch was also provided in the case it was required.

cwop

This option was provided as a vehicle by which the author could enter the code in which the courseware was being represented internally. It is considered a "back up" provision and would normally be removed as the authoring system matured.

cmds

This option allowed the author to redisplay the current menu of options. This was required, for example, when the menu area contained a display of the current font set while the author was placing text on the screen. It was the author's option as to how long the font set was visible.

run

The run option allowed the author to execute the courseware or a selected portion of the courseware to determine its adequacy.

Note: A number of useful options can be provided for the author during the execution of the courseware. For example, for experienced authors a step-by-step execution with a display of the internal code representing the course could be provided. Other options might include the display of variables and their contents. Problems can be encountered in the execution of a course if parameters for the execution are set outside of the domain of the code being executed. In this implementation the author could execute to any point of the courseware created, and mark the point at which newly created code could be inserted.

ed:it

The edit option allowed the author to execute a defined portion of the courseware and to have the internal code of the courseware placed on the screen. This code could be edited and immediately executed to determine if the change was appropriate.

Note: Editing can be a complex process especially when text and graphics are combined on one screen. When a screen may represent the effect of executing many different portions of the internal representation of the course, the problem of identifying for purposes of editing the source statements is a major problem.

he:lp

This feature allowed the author to obtain help in the use of options listed in a menu. In this case the help support was context dependent.

ex:it

This option signaled to the authoring system that the author wishes to leave the authoring process, and to start the processes associated with saving sufficient information so that authoring could be restarted at some later date.

14.3 A Typical Authoring Sequence

Using the commands of the main menu, a typical sequence of authoring steps would be as follows:

1) **label**	create a label to mark the beginning of the sequence
2) **prr**	define this point as a restart point
3) **screen**	set the desired screen format else the default will be a 45 column screen
4) **erase**	erase the whole screen
5) **picture**	display a drawing on the screen
6) **text**	enter text to the screen

7) **store**	store keywords to be used for answer judging
8) **input**	define the location for a keyboard response from the student
9) **trans**	define the transformation desired on the input
10) **answer**	carry out the type of answer analyses desired and appropriate replies
11) **tsb**	hold the screen until the student touches the spacebar or return key
12) **run** or **edit**	to observe exactly how the sequence will appear to the student

14.4 The Nature of the Author Interface

The nature of the interface is of course dependent to some extent upon the equipment and software in which the authoring system was written. This implementation was made on a DEC VAX 11/780 using DEC Gigi terminals with color display monitors. The terminal monitor provided a resolution of 767 pixels horizontally, and 240 vertically. Colors were limited with the Gigi terminal to blue, red, magenta, green, cyan, yellow, and white. The system was written in the Elf (Educational Language Facility) developed in the Division of Educational Research services. This language is highly typed, allows for the definition of complex data structures, and has the appearance of Backus Naur form. One desirable feature of Elf is the ease with which it can be used to parse any string of information. Changing the commands in the menu to another language such as French would be a trivial task.

From an author's point of view, the authoring system has the following surface appearance:

1) The author gets what he sees on the screen in all actions which create a screen effect. Thus the author creates visual effects as he would like the student to see them.

2) The system directs the author, through statements or questions, to provide the information it requires or to select a command from a menu.

3) In most cases the author's input is made through the keyboard; exceptions are when the cross-hairs are used to mark an area on the display screen.

4) Commands have a short and long form; these are evident in the menus displayed. Where possible commands are listed vertically to increase discriminability.

5) Redundancy of commands in different menus is purposely provided in order to increase efficiency of the authoring process. This reduces the frequency of movement among menus.

6) The menus provided in the first implementation are not true windows; they maintain a fixed location on the far right of the display screen, and or at the bottom of the display screen.

7) Menus form nodes in a tree structure in most instances; the depth is seldom greater than three.

7) Commands, help sequences, and error messages are context dependent.

8) Author input is checked; numeric values are checked for range, and alphabetic characters are checked to ensure they belong to the set currently appropriate.

9) Where an important commitment must be made by the author to a set of parameters, confirmation is requested and changes allowed before action is taken.

10) Throughout the system the author is provided with access to appropriate lists of information to reduce memory load.

11) Provision is made for continuity of information between authoring sessions.

12) All storage of information is made in "human readable" form to allow access to the data from more than just the authoring system, i.e., editors provided as part of the computer's operating system could be used.

13) As much as possible commands have a consistent meaning throughout the system.

14.5 Specific Functional Units; Problems and Recommendations

14.5.1 Screen Format

The author is questioned as to which of three screen formats is to be used. The three formats provided were a 45 column screen, a split screen the right side of which had 40 columns and the left side which was reserved for fixed-positioned graphics, and an 80 column screen. Different font sets were not used to provide for different character sizes required for the split screen and the 80 column screen. This was accomplished by the text size parameter of the DEC Regis graphics code used by the Gigi terminal.

The three screen formats provided adequate flexibility in creating screen displays with varying amounts of information. Although a "magnification" of characters could be carried out this was not implemented because the smoothness of the characters was lost. In addition there is little evidence to support a need for large sized characters except perhaps in the case of visually impaired students. Additional problems are encountered when characters of different sizes are rotated because the pixel resolution is not the same in the horizontal and vertical direction and this can result in some characters being almost unreadable. Although the Gigi terminal and its Regis graphics language provided for 16 different sizes of characters, in general the small sizes were too small and the large sizes where too large.

Since so many of the authoring processes cause screen effects, the type of screen format currently in use by the author must be made available through the software system. For example, when the area of the current screen is marked for keyboard input by the author, the system must know the screen format since a 10 character width under the 45 column screen format is not of equal length to a 10 character width under the 80 column screen.

Another major problem which has to be contended with is the font size used for the author directives and menus. This problem is caused by the need to switch back and forth between the font size associated with a given screen format as it is being created for the student, and the font format in which the author receives directives. For example, the font size in which author directives are given should remain the same regardless of whether the current screen is a 45 column or 80 column format, a regular font, or a symbol font set. Input and output to the author should be handled by a separate software subsystem.

14.5.2 Scale Option

Use of early versions of the authoring system indicated that an author would have difficulty aligning text vertically, and identifying line numbers unless there were some guides made available. In effect, this option "scales" the screen vertically placing dotted vertical lines at each 5 column position, and the digits 0 to 32 in steps of 2 to identify line positions.

14.5.3 The Pattern Option

Early versions of the authoring system did not have this option. This option was created in order to increase the efficiency of authoring and to reduce errors and inconsistencies in author moves which were highly algorithmic, as is the case in analyzing the adequacy of the student's answer. For example, in analyzing the adequacy of an answer the author might check a range such as 2.5 to 3.5, erase a portion of a screen, enter the text mode to give the student a confirming reply, do some simple arithmetic for score keeping, and perhaps specify a branch or take the system default branch. This pattern of moves would have to be repeated again for checking a series of wrong answer ranges, but with a different reply to the student, and with slightly different branching. In order to take advantage of the repetitiveness of the moves, the author can enable the pattern option and have all the commands used saved with the exception of those commands which are of a supportive nature. Commands of a supportive nature are commands such as "show", "help", etc. The pattern of commands can be saved with a user supplied name and comment, recalled for later use, and modified through a set of edit commands.

Although the pattern option was initially provided in the answer analysis section of the authoring system, its use was extended by making it available in the main menu. Thus, a pattern may be created of a major series of instructional moves such as might be required for the construction of many examination questions of the same general format.

The major functional units of the pattern option is illustrated in Figure 14.4.

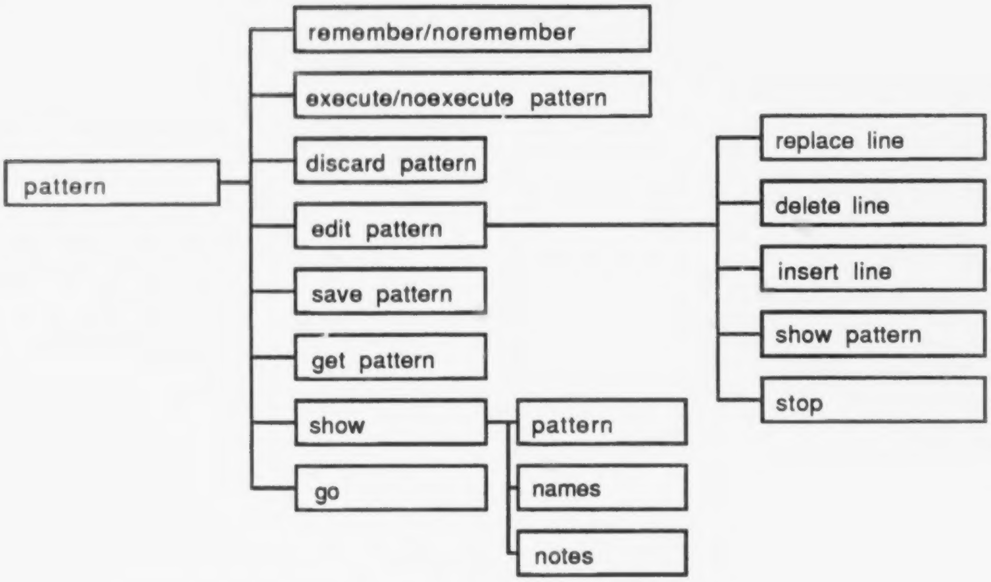


Figure 14.4 Functional Units of the Pattern Option

The options "remember" and "noremember" enable and disable respectively the pattern creation feature. A new pattern cannot be remembered if a current pattern is available. The current pattern must first be explicitly "discard"ed by the author. The option "save" is provided to save a pattern for latter use. A null pattern cannot be saved, and the author will be so informed if the attempt is made. The option "get" is followed by a request to the author to identify the pattern by its name. If the name is a legitimate pattern, the pattern is displayed to a maximum of 15 lines in a window. If the pattern is greater than 15 lines, touching the return key will display the next 15 lines, and so on. The option "show" allows the author to see the list of pattern names already created and their associated notes.

A pattern may be edited through the use of the "edit" option. The edit options allows the replacement, deletion, and insertion of lines. This is useful when a pattern is acceptable except for the positioning of the cursor or the beginning line of a screen erasure. The edit feature allows the author to make a simple change to an existing pattern and then to make use of it rather than creating a completely new pattern.

The execute option invokes the use of the current pattern. Upon leaving the pattern mode, the first command and all following commands of the pattern are acted upon until an input is required from the author, such as the entry of text. The execution of a pattern is terminated when the last line, "end_pattern" which is automatically inserted by the system, is encountered. The current implementation echoes to the screen each command as it is executed. The "noexecute" option was provided to allow the author to change his mind after turning execute on.

The development of the pattern option indicated that commands of the authoring system should be divisible into those which can be automatically executed through the pattern option, and those which are to be ignored, such as for example "help". A generalized form of storing these commands in separate lists would have been helpful had it been planned from the beginning. Although the author would likely indicate in the note associated with a stored pattern the context of its use, it is possible to enable the use of a pattern out of the context it which it was created. In such a case, the commands of the pattern would be inappropriate and result in a series of error messages. However, the system would not "crash". Future enhancements to the pattern option should include the automatic saving of the context in which it can be used. A check would then be made upon enabling a pattern as to whether it was being enabled in the appropriate context. Although the editor used to modify lines in the pattern is not a visual editor, the commands are so short, a complete line replacement is easily made. However, the editor which was written to make changes to the internal representation of the courseware is sufficiently general to accommodate editing a line of a pattern. This latter editor is callable from any location within the authoring system.

14.5.4 Label Option

An author must have some method of referencing a location in the courseware. In the case of this implementation this was provided by up to 6 alphanumeric characters which could include digits. Even if a hierarchical list structure is the basis for storing the courseware, some method must be available to the author to get to the desired node. Divisions of the courseware into units in this implementation was entirely up to the author, with the exception that when the author was starting to author a course for the first time. In the latter case, the first label in the course was the course name, the second label the word "start", and the third label was made up of the first three digits of the chapter number and 000. There were specific reasons for these three labels being created automatically, and these are related to the CAI run-time system under which the author produced courseware would be expected to run. The reasons were as follows:

1) In the run-time system, if a branch could not be made for some reason, a branch was defaulted to the label defined by the course name.

2) An instructor could always count on calling for an execution at the label "start", or if he wished to get quickly to the end of a chapter, to execute at the label "end". These were found to be useful conventions.

3) Because alphanumeric labels were allowed, in order to create labels of a numerical sequence by a default option, the authoring system had to keep track of the last numeric label. The third label noted above ensured that an appropriate starting point of such a numeric system always existed.

Except in trivial courseware, the number of labels could be quite large, i.e., hundreds. Thus a list of these labels is required with an accompanying note identify its nature (usually with a comment about the subject matter which follows) is required to assist the author.

Figure 14.5 illustrates the major components of the label option.

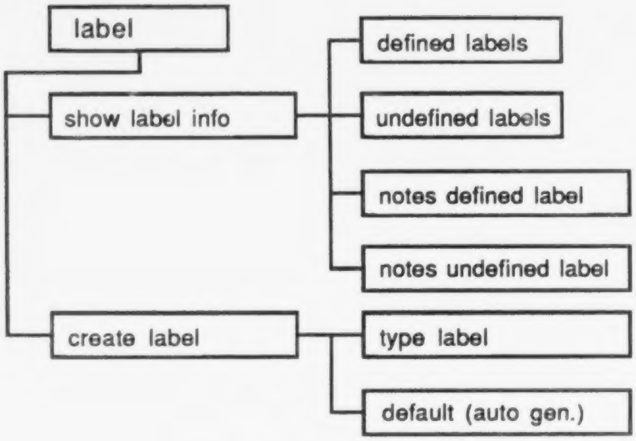


Figure 14.5 Functional Units of the Label Option

With reference to Figure 14.5 the label option has two major choices, to either show information about labels or to create labels. If the show option was taken, the system then asked the author to select from a menu whether labels to be shown were already defined, were used but their location undefined, to show notes associated with defined labels, or to show notes associated with undefined labels.

If the option taken was to create labels, the author was allowed to specify a particular label using the characters a-z, and 0-9, or to take the default which would create a unique numeric label. If the author decided to create a label, before this label would be accepted, it was checked against the current list of defined labels. If it was found to be a duplicate, it would be rejected and the author asked to create another or to take the default option. An accepted label was always displayed for the author.

The labels and their notes have to be stored to provide continuity from one authoring session to another. This implies that in order to establish the reference point for the generation of the numeric labels upon entry to an authoring session, the list of labels had to be searched from bottom to top until the last defined numeric label is found. This label then formed the reference point for the generation of further labels during the current authoring session.

Two lists of labels have to be kept. One list and its notes is needed for those labels which have been created. Another list has to be kept as a "pending label list", these are labels to which a branch has been defined, but the exact location of this label in the courseware has yet to be defined. When a new label is created, the pending list is checked for a match. If a match occurs, then the label is removed from the pending list.

There is little doubt that in this implementation of the authoring system the use of labels was a nuisance, and required the creation of software for making many checks. A hierarchical system would not completely remove the necessity for defining, for example, the point of execution, it would likely reduce the need for labels considerably.

14.5.5 The pr and prr Options

The pr option was used to control implicit branching, and under most circumstances the author would not be required to make use of the pr. However, the author must take into consideration the definition of where a student will be restarted in the courseware at the next session, i.e., make use of the prr option. Since the authoring system allowed the accumulation to screen displays, it does become important to restart the student at a point in the course which will ensure a screen to be complete, i.e., without missing components such as a drawing.

The use of the screen option to format the screen also has implications to the restart point. Immediately following a restart point the screen must be set to the appropriate format as the default format may not be appropriate at the next signon. There are also considerations which relate to the run-time system. The run time system has provisions to allow a student, upon signaling that instruction is to stop, to take four options: to stop, to go to a table of contents for the chapter, to continue, or to restart. A movement to a topic defined within the table of contents can only be made to a label, and the screen format at this point may be different than the format used for presentation of the table of contents.

The problems associated with a restart point are not present in authoring systems which restrict each observable screen display to be a finite entity, i.e., are strictly "frame oriented", nor is it a problem if such short sequence of courseware are created that the student can be required to always start at the beginning of a lesson not fully completed.

14.5.6 The Text Option

The text command is one of the more complex components of the authoring system because of the many variations and attributes which are available in placing text to the display screen. Major issues which had to be considered in the design of this option were as follows:

1) To what extent should this option provide features similar to a word processing system frequently used for desk top publishing? Features such as automatic centering of text, left-right justification, cut and paste, automatic pagination and indenting are some of the common features available.

   Primary consideration has to be given to the effect of such features on the learning of the students. It must also be kept in mind that creating instructional courseware is not the same as desk top publishing. We noted that even with the facilities of the text option provided in this implementation, there was a tendency for the author to become verbose because text could be easily placed on the display screen.

   An "uncluttered" screen is desirable and to the extent that the attributes of any text option promotes this, it is good. An uncluttered screen is one in which it is easy for the student to discern units which belong together, in many cases these units are conceptual units and not solely word units.

2) To what extent should the text option provide features characteristic of a visual editor?

3) There are instances where the formatting of text is required at run time. How should these instances be defined by the author?

4) Complex equations are difficult to define in such a manner as to provide for a pleasing and clear spatial relationship among elements. Should these be created as graphic-like entities, or be part of normal text definition?

The functional units provided for in this implementation are illustrated in Figure 14.6.

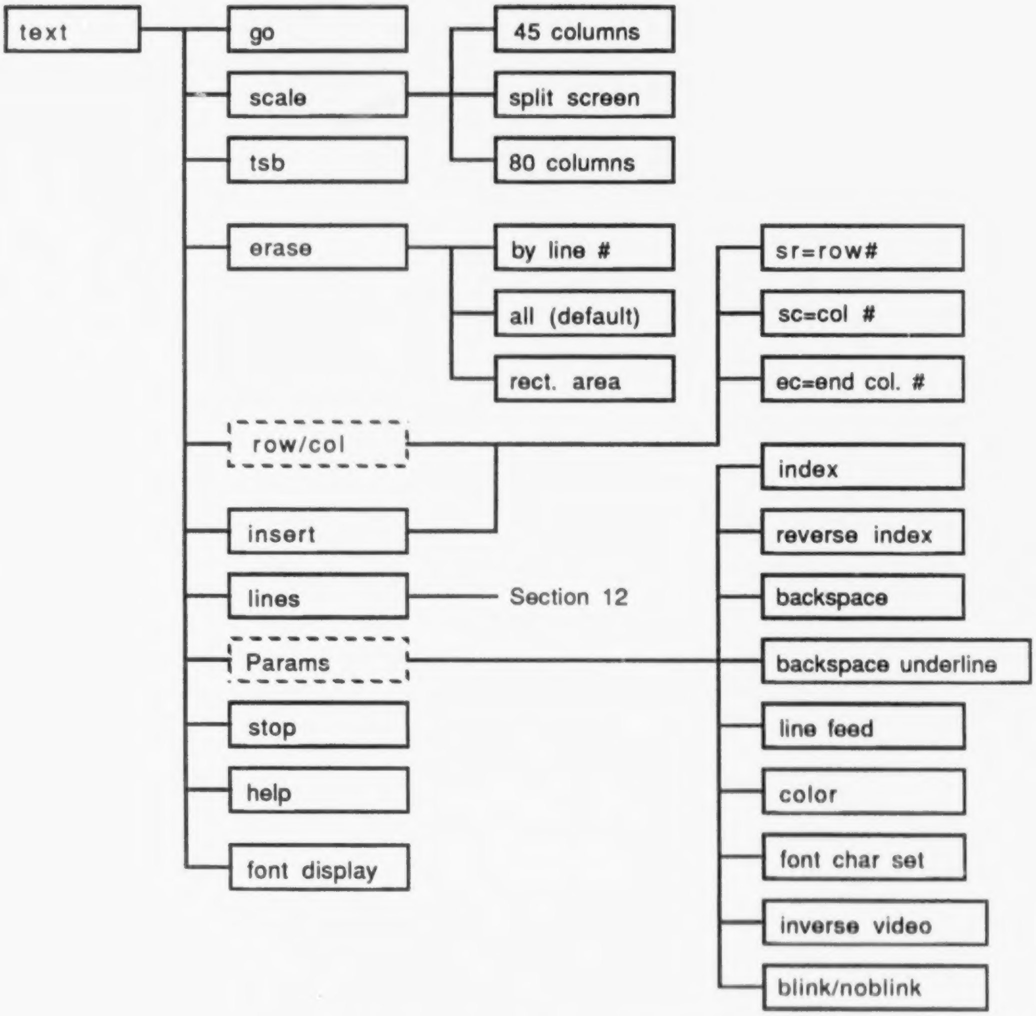

text
go
scale
tsb
erase
row/col
insert
lines
Params
stop
help
font display
45 columns
split screen
80 columns
by line #
all (default)
rect. area
Section 12
sr=row#
sc=col #
ec=end col. #
index
reverse index
backspace
backspace underline
line feed
color
font char set
inverse video
blink/noblink

Figure 14.6 Functional Units of the Text Option

The primary command menu under the option to create text on the screen was divided into three parts, although it appeared as a single menu. The options "scale", "tsbar", and "erase" are identical to the same commands in the main menu. These were included in this menu to save the author time in moving back to the main menu. The "fonts" option allowed the author to have the alternate font set displayed in the menu area when text was being placed on the screen. This was necessary since the alternate font set contained an additional 92 characters. These 92 characters were displayed beside the characters of the normal keyboard so that the author would know which keyboard character would produce the desired alternate font character. (Some systems provide the alternate font set but not in association with the current set, necessitating the author to "count the key positions" before the appropriate key is identified.

The "lines" option provided the author access to the creation of lines, boxes, and circles on the screen. The author could use these to highlight textual statements or to create pointers to draw the student's attention to a particular statement.

The "insert" option allowed the author to display the contents of a stored string or a variable. The option "comment" allowed the author to insert in the internal representation of the course, a comment. This was useful for identifying errors, such as a typographical error, which was to be corrected in the editor. The "cmds" option was provided to redisplay the menu, as the author could create a display containing text extending into the area of the menu. In addition this option was required when the author wished to replace the display of alternate character fonts with the primary menu.

The second set of parameters were concerned with initial positioning of the cursor to the required row and column, and to set the distance to the right of the starting position which would be erased. In the menu the starting row appeared as "sr=0", the starting column as "sc=0" and the parameter "ec=44" referred to the ending column to which an automatic erasure would be made. The default values for row and columns was zero, and the default value of the ending column depends upon the screen format used.

The third set of options provided the author with the capability of setting special attributes and special cursor positions. The following were implemented:

#i:index	index the cursor down # rows; used to create subscripts
#r:rvindex	index the cursor up # rows; used to create superscripts
#ba:backsp	backspace the cursor # character positions (allows for overstrike)
#bu:_____	backspaced # positions and underline to initial position
#n:linefeed	position cursor # rows lower and at left margin
cx:color	change to color "x"
#f:font	switch to font numbered #
{:invid :}	{ turns inverse video on; } turns inverse video off
B1:blink:B0	B1 turns "blink" on, B0 turns "blink" off

In this implementation a decision had to be made as to whether every character typed by the author would be monitored so that special function keys could be used. Because the environment was a terminal to host environment, it was felt that the speed would be too slow because of the amount of communication which would be required. This decision prevented the use of the keyboard's "arrow" keys for cursor positioning. A different environment would allow a more direct approach to cursor positioning. This did require, however, the use of an "escape" key in order to break out of the typing of text to the screen for setting additional parameters. In the current environment this prevented deleting and correcting characters beyond the point at which the escape key was used.

The setting of the row and column parameters was deemed after considerable experience, an easy way to set the cursor. The column position defined the left hand margin, making the use of two columns of text easy to create.

The creation of subscripts and superscripts was useful, and considered faster than the methods used on some micro computers. The backspace and underline was useful in the creation of equations.

Although the current set of operations provided for the author proved easy to use based upon the creation of approximately 18 hours (average student time to completion) of a course in matrix algebra, considerable improvements could be made. These improvements, to some extent would be contingent on the availability of a more technologically advanced authoring terminal.

14.5.6.1 Suggestions and Recommendations

1) The placement of text to the screen may be considered in the context of a specialized editor. That does not necessarily imply that all the features of a word processor are required, or are even desirable, particularly those features which would encourage the author to become verbose and in fact create a "textbook".

2) We consider the ability to have a freely formatted screen which at any point during execution may contain elements created sometime earlier, a useful feature for the author. This feature makes editing complicated since pointers must be maintained between the elements on the screen and the internal representation of the courseware.

3) The capability to place simple lines on the screen in the context of text is useful to the author. However, when these line cross lines of text, a problem of erasure is encountered as erasing a line may require reconstruction of the underlying text. The problems inherent in some micro systems which require the use of a bit map when graphics and text are intermixed should be avoided.

4) Some features used were found not to be very useful; in effect they were solutions looking for problems. The use of inverse video makes text difficult to read and better emphasis techniques can be used. The use of the blink feature although a little more useful than the inverse video, could be done without.

5) Using special keys, such as functions keys, would be useful for creation of special characters, superscripting and subscripting. The method employed in this implementation was considered faster and less prone to error than for example the techniques used in the MacIntosh of "dragging" the cursor over the character to be moved into a subscript position. In addition the creation of a superscript and a subscript in the same column position is easily achieved in the current implementation through the availability of a backspace operation.

6) The use of automatic repositioning of text with the deletion of characters is most useful, perhaps not so much during the text creation phase, but during the modification phase of courseware development. In our implementation the text contained special embedded control characters. If several words were removed from line n, the n+1th line was not automatically reformatted. Of course, this works well when characters and words are being removed, however, in the case of words being added, the additional space automatically generated beyond the last

line could interfere with text which is displayed several instructional steps in advance of the current position being edited.

7) We believe with the increased power of the text to screen operation, particularly with the use of more powerful editors, there is a tendency for authors creating material on the "fly" to forget about how much reading material is being created. It would be desirable to have some parameters which would indicate to the author the extensiveness of the text created, e.g., words per screen, screens per questions posed, etc.

8) Careful consideration needs to be given to the creation of complex equations on the screen. The availability to the author of a line drawing feature in the context of text definition allows for much easier construction of large square brackets around matrices, square roots, and lines between complex numerators and denominators. In order to achieve an acceptable spatial arrangement between elements of a complex equation, pixel positioning rather than character column positioning is required. Storing an equation as a graphic would allow the author to recall for display complex equations without going through the tedious task of their reconstruction.

14.5.7 Lines Option

The lines option, because it could be used with a screen display of text, proved quite useful. This option allowed the drawing of lines, circles and boxes using either a solid or dashed line. Additional attributes were "blink", complement, and color. The functional units of this option are illustrated in Figure 14.7.

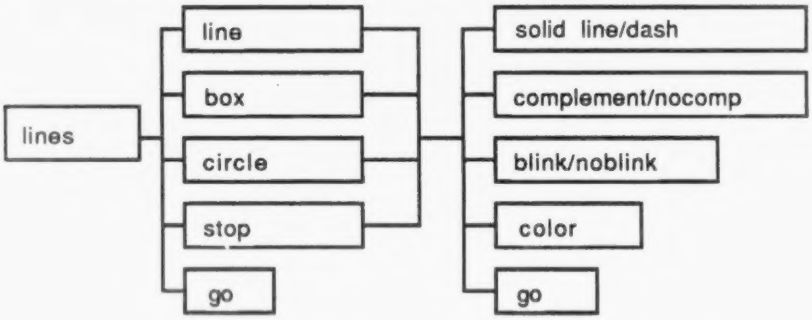


Figure 14.7 Functional Units of the Lines Option

In the current implementation the "lines" option used the cross-hairs of the authoring terminal to define the screen positions at which the drawings would be made. The "line" drawing feature was from point to point with as many points being used as the author wished. The "box" was constructed by first marking the upper left corner of the box and then the lower right corner of the box. Special markers identified the positions marked. The circle was created by first marking the center and then the radius.

Attributes associated with the lines option were set prior to the line option taking effect. Since no true windowing was being used, there was no problem of the author having to bring text "up front" after creating a box around it.

Major consideration has to be given to the editing of the lines in the context of text on the screen. This implementation did not include a true erase feature, although an erase could be effected by placing an identical line in compliment mode collinear with the line to be removed.

14.5.8 The Draw Option

The draw option allowed the author to create drawings "on the fly" just as it would in a classroom on a blackboard. However, in this implementation the provision was made for the automatic saving of the drawings created so that they could be used repeatedly as required in other parts of the courseware.

There are a number of issues which must be considered when designing an authoring system which includes of graphics:

1) How much power should the graphics software have? Some authors escalate their demands to the extent of wishing to be able to construct and rotate in 3 dimensions a molecular representation of DNA. This is laudable, but a practical limit has to be drawn at some point.

2) Because some drawing feature is available, should it necessarily be made available to an author? The graphic capabilities of some micro computers is impressive, but it does not follow that all courseware must make use of such graphics. Clearly, the instructional and subject matter context is important.

3) As with the use of text, access to a good graphics editor is very important. If editing of graphics is to permitted on an object on the screen with text, then careful consideration must be given to identifying the internal location of the object and the span over which editing can proceed. It is possible to have a single editor which is sufficiently context dependent to know of the data structures within which a graphic object or text is located.

4) To what extent will a drawing mode have the features available for creating text? This question is important when elements of a drawing require labeling. Such labeling may be simple words, or more complex mathematical expressions requiring special character fonts, superscripts and subscripts.

5) We question whether a single graphics software package can be expected to accommodate the needs of all authors and subject matter areas. The needs are too diverse.

6) It is likely that for many applications, a graphics pad or its equivalent is required to allow the construction of free-hand drawings.

The current implementation provided for the creation of graphics. There is not doubt, however, that the software created reflected the immediate needs of the course on which the authoring system was being tested and the software and hardware capability of the authoring terminal which was being used.

Figure 14.8 illustrates the functional components of the draw option which were implemented.

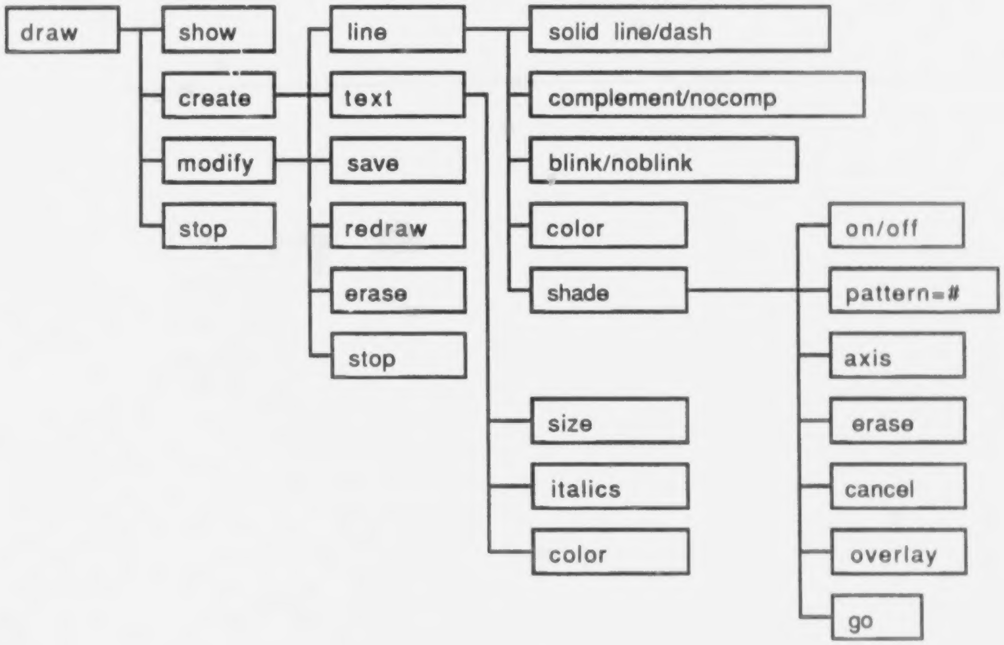

draw
show
create
modify
stop
line
text
save
redraw
erase
stop
solid line/dash
complement/nocomp
blink/noblink
color
shade
size
italics
color
on/off
pattern=#
axis
erase
cancel
overlay
go

Figure 14.8 Functional Components of the Draw Option

The author was allowed to construct two types of drawings. Both types were fully relocatable on the screen, i.e., all coordinates where relative to a single absolute coordinate which could be substituted during course execution. The graphic designed to occupy the right half of the screen during split screen format, during course execution was always positioned at one absolute screen location. In retrospect, this distinction was not necessary.

Experience suggested that courseware could have hundreds of drawings, therefore it was important that drawings receive some sort of identification and associated note before becoming part of a graphics library. The "show" option provided for the display of the names of existing drawings which the author had created, and by a further option the comments associated with the drawing could also be displayed.

The current implementation as indicated by the first menu of the draw option, provided only for the drawing of straight lines and does not easily accommodate irregular curves and circles. The text option provided for labeling of any component of a drawing and had positioning at the pixel level of the screen. The "redraw" feature allowed the author to observe the drawing as the student would, with all the target marks used during the creation phase removed.

In using the draw feature it was found that efficiency could be increased by starting with an existing drawing and making additions to it. For this purpose the "modify" option was created.

The "save" option was used by the author if he was satisfied with the drawing and wished it to be inserted into the courseware. The erase option was a call to the erase feature described earlier as part of the main menu and the text option. It provided for screen erasures.

Once the author selected the line option, a menu of attributes was presented. These attributes where identical to those provided in the "line" option discussed earlier, however, one additional option called "shade" was made available. This option allowed closed figures to be shaded with a solid color or a colored pattern. Four patterns were provided. Experience with the "shade" option found it to be too obtuse to use effectively. This was due to the terminal characteristics which required that shading be defined from a line being drawn to a defined y axis. The capability of simply identifying the closed figure which is to be shaded is much preferred.

If the "text" option was selected, a menu which allowed the specification of the "size" of the text, the angular slope for "italic" characters, and "color" was made available. In each case, typing the option was followed by asking the author to specify the parameter from a defined set.

The major deficiency in the implementation of the option to create graphics was that no graphics editor was created to be used within the context of the authoring system.

14.5.8.1 Suggestions and Recommendations

1) Consideration should be given to the use by the authoring system of different graphics packages, each designed with common and unique characteristics which are best suited to the nature of the subject matter being developed. This implies that such packages must be capable of being integrated within the context of the authoring system.

2) It is desirable to have a size change option.

3) It is frequently useful for an author to be able to make a "test drawing" to confirm whether a concept can be illustrated clearly. Although this feature can be carried out with a "test chapter" it is useful within the context of the chapter being created.

4) Labeling of drawings should have the same text attributes as the text option used for placing text on the screen, e.g., font changes, size changes, color attributes, and superscripting and subscripting.

5) The "show" option used to display information about the drawings created, requires an additional feature which would allow the searching of the list for keywords. This feature raises the question as to what information should be stored in a graphics library.

14.5.9 The Plot Option

The plot option provides the author with the capability to create four types of plots: bar graphs, a plot of individual points, continuous line graphs, and vectors. In creating the software for the creation of these types of plots, the following factors were considered important:

1) The author should be able to create plots from simple or complex data.

2) The graphs should exhibit clarity in their representation; they should not be cluttered with useless characteristics.

3) The plot should be an accurate reflection of the data used to create the plot, but need not show the absolute values of the numbers used to make the plots.

4) The author should be able to embody in the graph instructional features which are not normally part of graphical representation of data.

In designing the capabilities of the plotting software, consideration was given to the kinds of plots an instructor would use in the classroom or desire in a textbook. The following are some examples of such illustrations:

a) A function is illustrated without reference to specific values marked on the X or Y axis; values on the axis might in fact be alphabetic characters or names.

b) Specific data points in a graph are projected back to their corresponding X and Y axis values.

c) The value of integral units on the X and Y axis are identified with simple integers which could be quite different than the data which was actually used to plot the graph.

d) Axis may take on many different configurations depending upon the quadrant required for the representation of the data.

e) Text is frequently added to the graph for explanatory or attention purposes not just for labeling purposes.

f) More than one graph may be plotted using the same axis. Sometimes a set of graphs is required to illustrate a concept.

g) The number of units along each axis may be quite arbitrary.

Such characteristics as noted above are not part of most plotting packages which produce graphs typically seen in technical publications.

Figure 14.9 illustrates the functional units of the plotting facility available to the author:

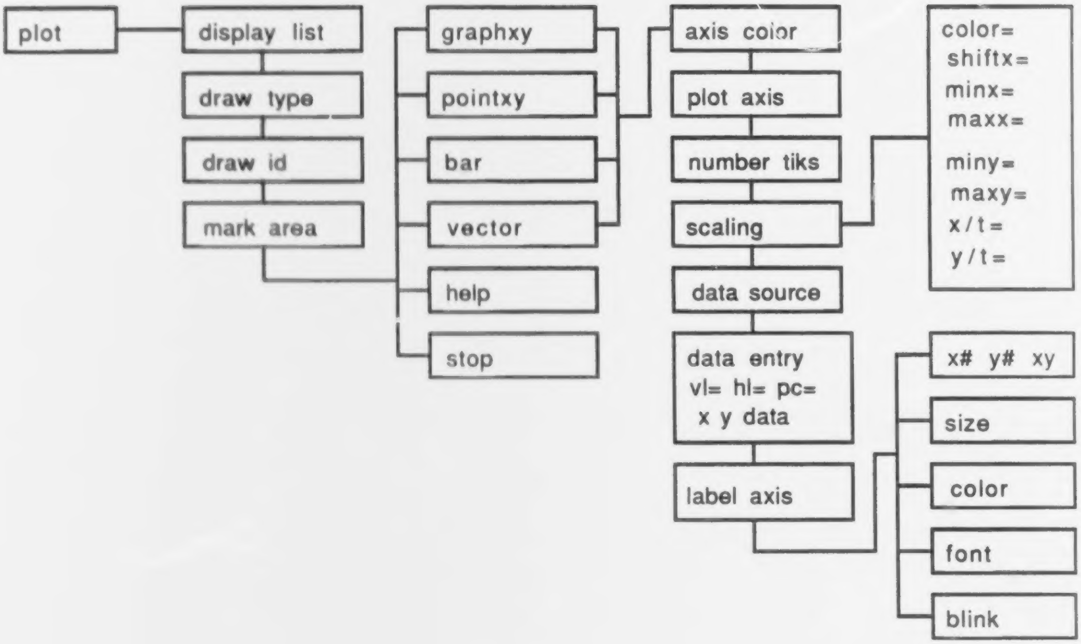

plot
display list
draw type
draw id
mark area
graphxy
pointxy
bar
vector
help
stop
axis color
plot axis
number tiks
scaling
data source
data entry
vl= hl= pc=
x y data
label axis
color=
shiftx=
minx=
maxx=
miny=
maxy=
x/t=
y/t=
x# y# xy
size
color
font
blink

Figure 14.9 Functional Units of the Plot Option

Plots created by the author are stored in the same manner as drawings, and include an identical type of identification. Therefore, the ids of plots and the associated notes were placed in the same type of list structure as other type of graphics. The list of all drawings and associated notes could be recalled by the author before starting the plotting procedure.

The first step in the graphing process was a request of the author to use the cross hairs to mark on the screen the upper left position which the graph would occupy. (A lower right position was not marked as the drawing could be made any distance to the right and below the initial point marked). From a menu the author is then asked to select the type of plot to be created, e.g., graphxy, pointxy, bar, vector.

The plotting sequence was divided into the following steps:

1) **Plotting the axis:** The author was requested to select the color to be used for the axes and then to use the cross hairs and mark the upper and lower point of the Y axis. The author is then requested to define in a similar manner the X axis. The only constraint on the definition of the axes was that they must touch or cross each other. If this condition was not met the author was asked to redraw the X axis to meet the condition.

2) **Marking the Axes:** Next, the author is asked to type an integer indicating the number of units into which the Y axis was to be divided ("tik marks"). The number of units along the X axis was obtained in the same way. The author could stop at this point which would provide a marked axis for use in the course.

3) **Scaling the Axes:** The author is then asked to provide the color in which the plot will be made and the scaling information for the data to be plotted. The scaling information in the current implementation could be provided by the author in two forms: as minimum and maximum values for both x and y variables, or as the value of each marked unit on the x and y axis. After experience with the first form of providing the scaling information, it was found that a simpler and more useful procedure for many plots would be provided by the second procedure. In the case of the first procedure the unit values on the X and Y axis could take on unwieldy fractional values and multiples thereof, e.g., 1.66, 2(1.66), 3(1.6), and so on, hardly the type of unit an instructor would want for purposes of illustrating a concept. The current implementation holds both methods of scaling, such that if one method is used, the other is not.

4) **Data Entry:** Next the author is asked to identify the source of the data to be plotted. The options provided are keyboard or a file; if a file contains the data, the name of the file is requested. (In the case of file input appropriate messages are provided the author if the file cannot be found or opened for some reason). Where the plot could be simply created using integer values, the author enters these directly from the keyboard. However, if the function to be plotted is to be represented by plotting, say, one hundred or more points in order to achieve a smooth curve, entry from the keyboard of all these points would be unacceptable, and file input would be used.

5) **Data Points:** When entry is made from the keyboard, each point is plotted immediately. In addition, the author may specify four additional parameters for any data point:

```
vl={on,off}
hl={on,off}
color=
pc=x
```

For a point entered, "vl=on" would cause a vertical line to be dropped from the point plotted to the X axis; "hl=on" would cause a horizontal line to be placed from the point to the Y axis; "color=" followed by the color would set the color for the point; "pc=" followed by a character would cause that character to be plotted in the pointxy type graph.

6) **Labeling the Plot:** The last step in the plotting procedure, was at the option of the author, to label the graph. Four attributes were provided (size, color, font set to be used, and blink), and three labeling procedures. The following labeling procedures were created to provide for ease and efficiency:

x # =	this option allowed the labeling of individual "tik" marks on the X axis. For example, x5=B would cause the letter B to be placed under the fifth tik mark of the X axis.
y # =	this option had the same effect as x# but for the Y axis
xy	is used to place text anywhere within a pixel position on the graph. This feature was necessary to allow the author to create such things as a note explaining a particular feature of the graph. The author was presented with a set of cross hairs to locate the starting position of the text.

7) **Repeat Option:** At this point the author was queried as to whether more graphs were to be made. If the response was affirmative, a further query was used to specify whether the last axes was to be used, or whether a new set of axes were required. From this point the procedure of plotting repeated itself.

14.5.9.1 Suggestions and Recommendations

For the same reasons that a word processor is not that which is required for the creation of text displays, the usual plotting package is not that which is required to handle the plotting needs of the author. Such packages were not designed with instruction in mind.

Features which should be taken into account when software is designed for plotting within the authoring environment are as follows:

1) For many instructional functions, simple graphs are preferred to illustrate a concept. Such graphs can be constructed on the"fly" using integers data .

2) Where complex functions with many data points are to be plotted, these data points should be in a separate file and properly created using mathematical algorithms to ensure fidelity of form.

3) As with all graphics for instructional purposes, "pizzaz" or flashiness is to be avoided. Clarity of the concept being illustrated is the important feature.

4) Flexibility in labeling axes of the graph as well as elements of the graph plotted are required by the author.

5) It is useful to be able to plot 3 dimension graphs and to allow the author to rotate the axis to a desired position, and then to "snap" this perspective for storage and later display within the courseware.

14.5.10 The Picture Option

The "picture" option allowed the author to display a drawing or plot on the screen for the student. The author is questioned as to whether the list of drawings (including plots) is to be displayed. This feature allowed the author easier recall of the drawing desired. If the drawing was fully relocatable, the author was asked to provide the id of the drawing and mark the upper left position of the screen corresponding to where the upper left position of the drawing would be displayed. If the drawing was not relocatable, i.e., it is a drawing to be placed on the right half of the split screen, the marking was not required. If a drawing could not be identified by the id given, i.e., a file corresponding to the drawing was not found, an appropriate message was give.

In the case of the drawings designed to appear under the split screen format, the author was queried as to whether a "new" drawing or the "last" drawing was to be displayed, or whether a drawing was to be "removed". The display of a drawing under the split screen format simulated the display of a drawing from an image projector or video disk, and these latter options followed the opening and closing of a shutter.

14.5.10.1 Suggestions and Recommendations

Experience with this option suggests that the need for a fixed position drawing for display under the split screen format was unnecessary. All drawings should be fully relocatable and their display and erasure should be handled in a common manner.

An additional feature which should be added is to allow the author to display a drawing without a commitment to have that particular drawing included in the courseware. Such a feature would allow the author to review a set of potential drawings, and after selecting the most appropriate one for the instructional function at hand, make a commitment to its use.

14.5.11 The Erase Option

The erase option provided the author with the facility to erase the whole screen or selected components of the screen. The one instance in which the erase was not used to directly remove something from the screen was when a drawing under the split screen mode was to removed. In the latter case the option "remove" was used and was analogous to closing a "shutter" on a projector or removing an image presented by a video disk.

Three options were deemed necessary under the screen erase:

1) erase the entire screen

2) erase the entire width of the screen but for a selected number of lines

3) erase only a marked rectangular area of the screen

Although the third option could have been used to cover the first two, the marking of the screen was considered too inefficient given that the first two cases are required far more frequently than the latter case.

In the case of erasing a rectangular area of the screen, the author was asked to use the cross hairs to mark the upper left and lower right of the area to be erased. In the case of erasing selected rows, the author was asked to give the beginning and ending row number of the rows to be erased. These values were checked to see whether they were within the boundaries of the number of rows available on the screen. A default condition was used to achieve the full erasure of the screen, i.e., touching the return key caused the whole screen to be erased.

14.5.11.1 Suggestions and Recommendations

An error which caused an erasure of the entire screen could not be immediately corrected because the screen at any particular time during the authoring process could not be recreated. Again, this is a problem in the manner the internal representation of the course was stored. The error could be corrected under edit mode, however this is not entirely satisfactory. The erase of the complete screen removed from view the material upon which the author would depend for maintaining continuity in the authoring process.

14.5.12 The Store Option

The store option went through a number of modifications as more experience was gained with the authoring system. The original provision of a fixed number of buffers or storage areas for strings containing such things as keywords used in answer analysis, and required by the interpreter, was inadequate and therefore extended. Use of the authoring system indicated that an author would use certain strings repeatedly for common instructional moves. For example, the author would ask the student whether a review was required or not. The key word "rev" or "review" had to be stored too frequently. The extension made was to retain the original storage of strings in buffers as required by the interpreter, but to allow the author to create an unlimited "bank" of strings which could be named, recalled for display, and loaded into buffers as required by the interpreter.

Since an author would interpret the word "store" as being applicable to the storage of any useful element, the author was first queried as to whether keywords, labels, or integers were to be stored. If integers were to be stored, the author was directed to use the "arith" option since it was assumed that the integers were to be stored as numerical values and not characters.

Figure 14.10 illustrates the functional units of the store option.

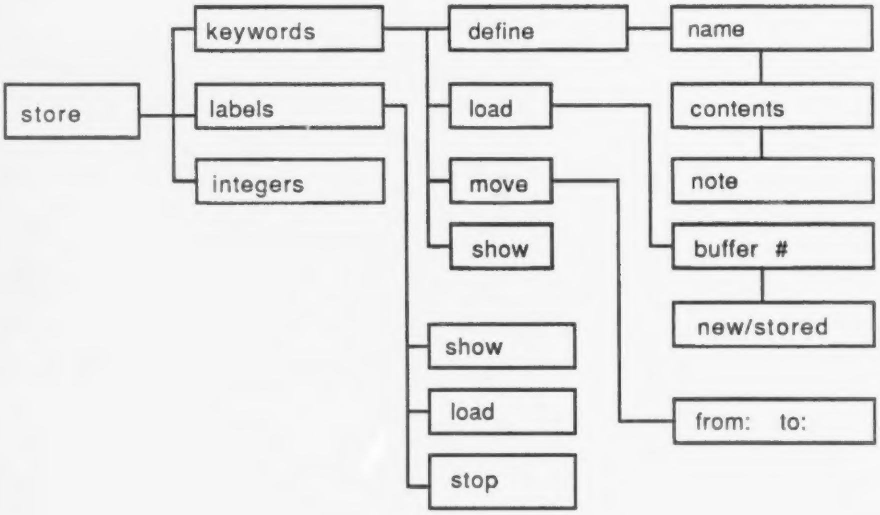


Figure 14.10 Functional Units of the Store Option

The author is presented with a menu with the options "define", "load", "move", "show", and "stop". The option "define" was used when an author wished to define a string and have it saved under a user given name. Typically, the author would store frequently used strings, such as "yes", "no", the string "a b c d e " for use in multiple choice responses, "review" and "continue", and commonly used key words. Since the author could name these strings, provision had to be made to store these strings and their names, and make them available for review by the author. The author could review these names by the option "show".

The option "load" allowed the author to enter a "new" string into a buffer as required by the courseware interpreter, but he could also load the buffer from the user defined bank of "store"d strings. Once a buffer had been loaded, the authoring system has to keep track of the fact, since the author may count on this being loaded at subsequent authoring sessions.

The courseware interpreter had provisions for moving one string from a named buffer to another buffer. The "move" option provided this facility. In addition, the interpreter had provisions for indirect branching, i.e., branching to a label pre-stored in a special register. This provision was made available under the labels option of the first menu displayed to the author. However, because there could be hundreds of labels, the author was given access to review the list of labels.

14.5.12.1 Suggestions and Recommendations

Although the storage of strings and labels had been expanded as experienced was gained with the authoring system, there are a number of provisions which would be desirable:

1) The author should be able to use the storage facility developed under this option for the storage of strings which are created in the definition of a complex equation. This string could then be displayed under the "text" "insert" option. The author could still do this under the facilities provided, however, the string would have to be defined without reference to its representation on the screen, and if there were many position control characters in the string, the chances of the equation being properly represented on the screen would be very small indeed.

2. If an author is to store a label, and if the author is given the provision to create alphanumeric labels (which we do not recommend) then the possibility exists that the label "stop" and the command "stop" will both cause exit from the option. This illustrates another reason for system generated labels. (The problem is not solved by selecting a different word such as quit, since "quit" could be defined as a label also).

3. Although the author could view the words and characters placed into storage, these were displayed as a vertical list without marking the blank characters (spaces). Since it is possible that some answer judging algorithms may depend upon exact character positioning, an exact representation of the stored string with character positions marked would be desirable within the authoring system. (This provision was available to the author, but only upon backing out of the authoring system).

14.5.13 The Input Option

The input option allowed the author to define a keyboard or pointing response to be called for from the student. Although a time parameter was available under the interpreter, this parameter was defaulted to its maximum value. If the author indicated that a keyboard response was required, the cross hairs were positioned on the screen and the author was asked to mark a rectangular area within which the student's response would appear. From the screen coordinates provided by the marking procedure, the authoring system calculated the number of characters which would fit this area given the current screen format. The author was then asked to indicate the maximum number of characters expected, and if this was not within the boundary of the defined area, the number of characters was set to that defined by the rectangular area. The author was then asked to confirm the definition of the input; if it was not confirmed the definition was restarted.

A response identifier was associated with all requests for input from the student. This response identifier is used to relate the student's response as stored in a performance recording file, to the location in the course where the input (usually a question) was asked. In this implementation the response identifier was automatically created by using the last label and the number of the line following the label in which the input was requested based upon the internal storage representation of the courseware.

14.5.13.1 Suggestions and Recommendations

Although the above definition of input by the author was quite adequate, easy, and efficient to use, a number of issues are still of concern:

1) Although a response identifier was automatically generated, the editing process could position a question in some different location in the internal representation of the course, thus the identifier would no longer be accurate. Although this would not lead to serious problems, it is deemed inappropriate. Thus, any editing procedure must take into account the possible modification of such identifiers.

2) The question as to how much identifying information is to be associated with an input must be posed. In this implementation a response identifier of "zz" indicated that if performance recordings were being taken during courseware execution, at this point no recording should be made. This might be the case where a question had been posed but it was one which would allow the student to review or continue. Under provisions of the current interpreter, if the response identifier was of the form "exam#", then this indicated to the system that a specific variable containing the results of an exam were to written to a special examinations mark file. This feature now suggests that the response identifier is more than a definition of the location of the input in the courseware. Further examples can be given. There may be need by the author to identify the response as belonging to a particular category within some taxonomy, e.g., a synthesis question which would be useful in the context of the question being answered correctly or incorrectly. The nature of the information associated within the input needs further examination.

14.5.14 The Transform Option

The "transform" option was designed to provide the author with a set of transformations which may be placed upon the student's response prior to the response being subject to an answer analysis process. For example, the author may wish to have all leading blanks, and all sub-strings which have 2 or more contiguous blanks replaced with one blank. The author may wish to remove certain words from the input if they exist, e.g., "a", "the" and the like.

In this implementation the author was allowed to define a case shift, and to replace a sub-string of the input with another string. If a "Boolean" type answer analysis was to be used, a special transformation was carried out which replaced the student's input with the rank order value of the keywords which was defined in a standards string and already stored.

14.5.14.1 Suggestions and Recommendations

The concept of defining transformations to be placed upon the student's input is considered important and requires further exploration as there are likely many transformations, some specific to a subject matter area, which might be defined. It is probably desirable for the author to be able to name and enable such transformation at the beginning, or at any time during the authoring process.

14.5.15 The Answer Option

This option is one of the more complex options available in the authoring system. From the author's and student's point of view, the processes checking the nature of a response must behave in an intelligent manner. Unfortunately, a student's keyboard response can be one of an infinite subset of responses, and it is up to the processes in the answer analysis, or answer judging facility to uniquely identify the current input. A trivial approach to the problem taken by some authoring systems is to restrict the input to multiple choice responses. The complexity of this option is due primarily to the processes required to abstract meaning from the input, however, within the domain of this option almost all the other processes of the authoring system may also be required.

There is also the problem for the author in keeping track of the processes he has already invoked as there may be in some instances, many involved steps in identify the nature of an input from the keyboard. The process of identifying a pointing-type response, is however, straight forward.

Figure 14.11 illustrates the functional units which were implemented.

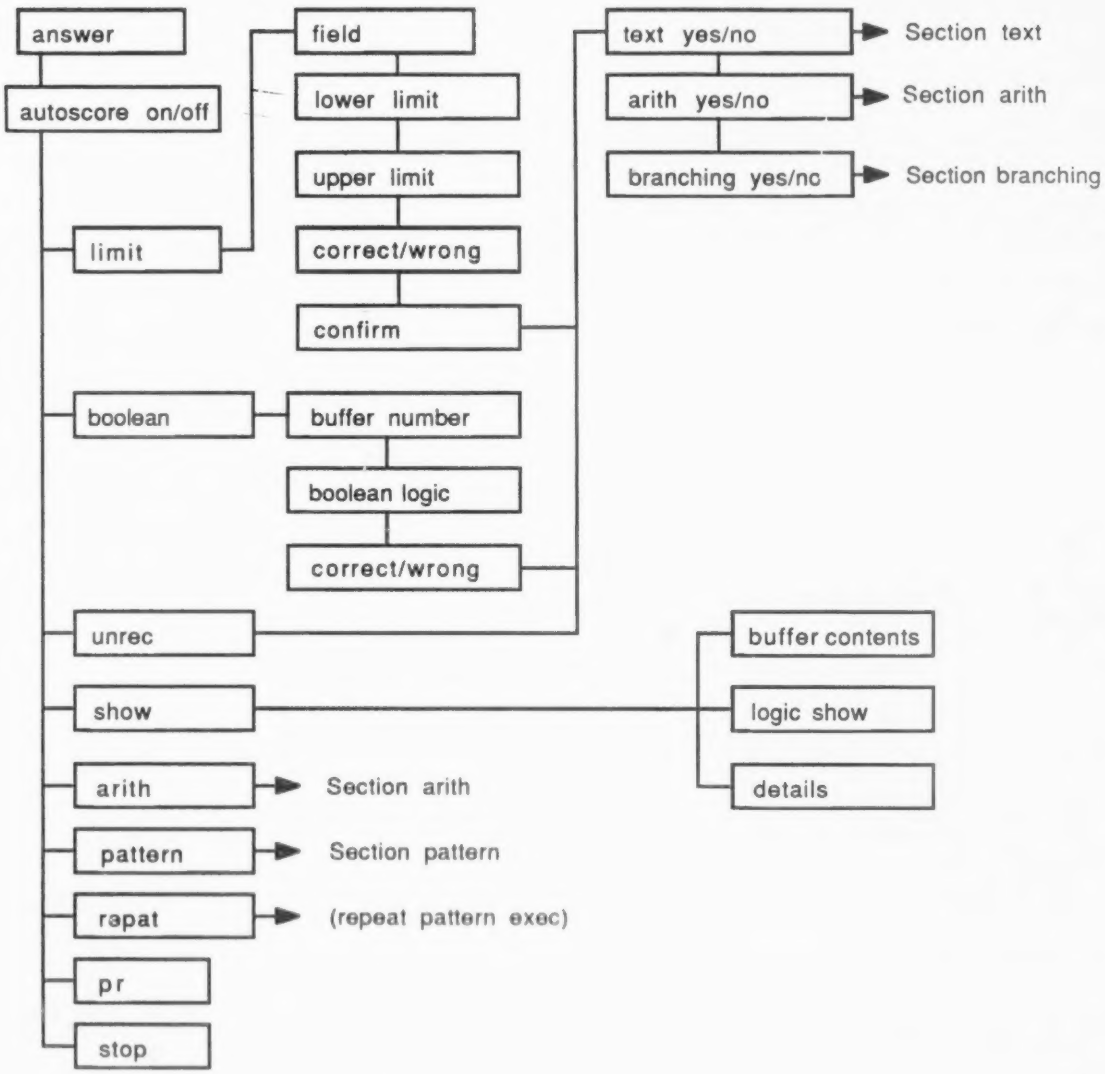

answer
autoscore on/off
limit
field
lower limit
upper limit
correct/wrong
confirm
text yes/no
Section text
arith yes/no
Section arith
branching yes/no
Section branching
boolean
buffer number
boolean logic
correct/wrong
unrec
show
buffer contents
logic show
details
arith
Section arith
pattern
Section pattern
repat
(repeat pattern exec)
pr
stop

Figure 14.11 Functional Units of the Answer Option

After initial experience with this option, the "autoscore" feature was added as it was found that authors were avoiding the use of the "arith" option to increment score keeping variables. When the scorekeeping option is enabled, 3 global system variables are created to accumulate the number of correct, wrong, and unrecognized responses identified by the answer option. A "clear" option to reset these 3 variables to 0 was also provided.

The two primary processes available under the answer option were a numeric range checking process and a Boolean keyword process. The numeric range (limit) handled, without author definition, both integer and real value input of the student.

Since the definition by the author of the answer analysis process was repetitive, the "pattern" option (templating) was made available so that once a process had been defined, its pattern could be used repeatedly with the need to only specify the unique features of the process.

The first query to the author was whether the analysis was for a keyboard or pointing response. The description which follows is for the analysis of a keyboard response.

The first menu provided the author contained the following options:

limit	to check a numerical range
Boolean	to do a keyword check on the response
unrec	to handle an unrecognized response
arith	to carry out any arithmetic operations
pat	to invoke the pattern creation feature (template creation)
repat	to re-execute the current pattern
pr	to control implicit branching (not normally required)
show	to display to the author the logic and associated notes

If the "limit" option was selected the author was queried for the position (field) of the number to be examined, the lower limit and the upper limit , and if the condition was matched whether the answer was to be considered correct or wrong. (Parenthetically, it should be noted that "correct" and "wrong", although producing different branching characteristics, are really determined by the nature of the reply the author constructs for the student. Thus, the terms defined only the implicit branching characteristics). The options selected were then presented to the author for confirmation, and if confirmed as appropriate the author was then queried as to whether he wished to construct a textual reply. If a reply was desired, the "text" option as described earlier was invoked. Upon leaving the "text" option the author was asked to give a brief description to which the reply was directed, e.g., inverse is correct. The author was next asked whether any arithmetic operations were required, if the answer was yes, the "arith" option was invoked. Following this, the author was asked whether any special branching was required, and if it was, the "branch" option was invoked. At this point the author was returned to the menu items noted above, and could proceed with another limit check, or to invoke the Boolean process.

The second answer analysis process was a Boolean approach to the analysis of the student's input. Upon entering this process the author was asked to provide the buffer which contained the keywords to be sought in the student's input. The contents of this buffer where then displayed and indexed with 1 to the number of keywords. The author was then shown another menu which provided guidance on how to create a Boolean statement using the elements of the buffer displayed. This menu contained the following guiding formats:

(#)&(#)=and
(#)|(#)=or
('#)=exact
(^#)=not
group=(#)

# represented the index number of the keywords in the currently displayed list of keywords

Thus, if the currently displayed keywords were (1) Clgry, (2) Albta, (3)Edmntn, and (4) Cmrs, to check for "Clgry" and "Albta" and not the presence of "Edmntn" the author would construct the following statement:

(1&2)&(^3)

An earlier implementation of this definition required the author to specify the relationship as

(Clgry & Albta)&(^Edmntn)

which was thought to be a more meaningful statement, however, it was found that authors made too many errors in typing, i.e., they would not type exactly the keyword which was in the keyword list. Since a search was made of the keyword typed by the author against the list of keywords, an error message indicating the typed keyword was not in the list would be given. The author would then have to redefine the standard. Errors associated with the use of the grouping character i.e., the (and) were identified in the parsing process used to identify the author's input. The position where the parsing failed was identified for the author. The author was then asked to identify whether the standard given should be handled with the "correct" implicit branching, or the "wrong" implicit branching, and then directed towards defining a reply, arithmetic operations, etc. as indicated in Figure 14.11.

The "unrec" option allowed the author to provide one or more messages with each failure to identify a student's input under the limit or Boolean functions.

The "show" option allowed the author to observe the contents of any buffer holding keywords, although the appropriate buffer was automatically displayed when the Boolean logic was being specified, as well as a flow chart representing the logic of the answer analysis process created. Each answer class was automatically given a unique identifier, and this identifier appeared in the flow chart. The author could also recall the comments associated with each attempt to identify the student's input.

The "pat" and"repat" options provided access to the pattern creation facility and execution of a pattern respectively, and as described earlier. Thus an author could make one pass at defining an answer checking process, and then use it repeatedly by having to create only the unique components of the process.

For a pointing response, the author was asked to use the cross hairs and mark a rectangular area on the screen, and then to identify whether this area, if pointed to, would generate the implicit logic of a correct or wrong answer. The author was then given the choices allowed following the limit and Boolean options.

14.5.15.1 Suggestions and Recommendations

In general, the "answer" option worked well, however, it does not reflect the full power which should be available in an authoring system and CAI delivery system. The kinds of facilities noted in Chapter 10 require further research and implementation.

1) Although the numerical limit and Boolean options worked well, and could cover a wide range of answer checking, in some cases they were obtuse to use.

2) The nature of the answer analysis processes relate to the nature of the input processes to some extent. For example, if the student is allowed to use a true subscript or Greek symbol as part of the input, this must be reflected in the options available to the author for identifying such input.

3) The author should be able to immediately test whether the process he has specified will in fact identify the input as anticipated.

4) The "intelligence" of a CAI system from the student's point of view depends very much upon the appropriateness of the system's reply to the student's input. Therefore, specific process perhaps unique to given subject matter areas are in need of development. For example, the evaluation of a numerical function is quite important to the teaching of mathematics, but not a foreign language. A diagnostic feature is not possible without having many different processes available for the author. However, the processes are dependent upon an integration of specifying the standards, transformation of the standards as well as the student's input, and the matching processes. These must be treated as an interrelated set of processes.

14.5.16 The Arithmetic Option

The "arith" provided the author with the capability of defining simple arithmetic expressions. To a great extent, the form of expressing the arithmetic options was constrained by the arithmetic capabilities of the interpreter which was restricted to the use of a fixed number of integer variables.

Figure 14.12 illustrates the functional units of the arithmetic option.

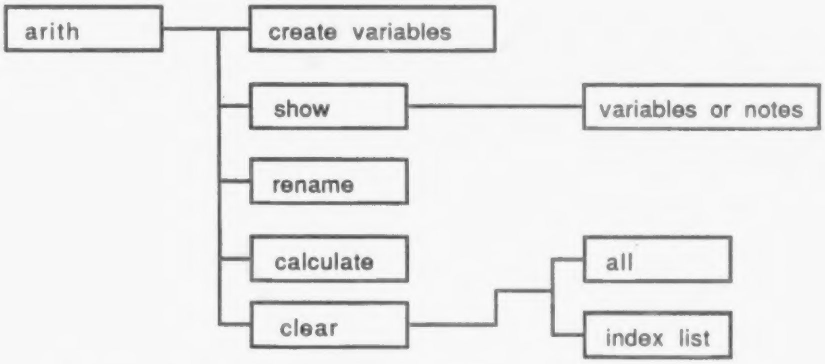


Figure 14.12 Functional Units of the Arithmetic Option

The author is presented with a menu providing the options "create", "show", "rename", "calc", and "clear". The create option allowed the author to define a variable by assigning it a user name, and a comment as to its use. The "show" option allowed the author to review the list of variables and associated notes. The "rename" option allowed the author to rename an existing variable. The "calc" option allowed the author to specify the arithmetic statement, while the "clear" option allowed the author to set either all variables to zero or to specify one or more as a set to be cleared to zero.

For purposes of score keeping, i.e., counting the number of correct, wrong, and unrecognized responses made by the student, the authoring system set up three special variables which could also be manipulated arithmetically. However, these variables could not be renamed by the author.

14.5.16.1 Suggestions and Recommendations

The arithmetic option was adequate for purposes of accounting within the courseware, but would not be considered adequate as a basis of providing arithmetic power useful in the generation of components of the courseware. Although a reverse Polish parser for evaluating arithmetic functions had been developed, it had not yet been introduced into the interpreter, and therefore its capabilities were not made available within the authoring system.

There are four considerations which should be taken into account in determining how much, and in what form arithmetic capabilities should be provided:

1) How much power is available or needed within the CAI run-time system and the courseware interpreter or compiler?

2) How much power is needed by the author for use within the authoring system in support of the creation of courseware?

3) What should the form be (syntax and semantics) of the calculating facility?

4) How much arithmetic power is required in the hands of the student?

With regard to the first question, it does not make much sense to provide arithmetic capabilities to the author for use in courseware when the courseware interpreter or compiler does not have the capability of handling it. Therefore, the capabilities of the authoring system software must be related in a minimal way to the requirements of the run-time system. Different subject matter will call upon differing amounts of computing power. The arithmetic power required for a course in linear algebra, calculus, and physics is considerably more than required for grade three arithmetic.

The author may be given more power than that required within the courseware if that power is required for the *creation* of the courseware. For example, the author may wish to have 3 dimensional graphs plotted and rotated to a specific perspective, and then have this perspective made available for display to the student. The author may wish to plot a sine function and have the results display to the student, but not actually calculate the sine function during courseware execution, would be another example.

The form of the arithmetic power in terms of the nature of the syntax and semantics provided is open to debate as are the applications of most languages which have similar capabilities. Some authors would argue for a Basic-like form, others for a Fortran, and still others for an APL-type notation. Some authors might even wish to have access to computing power for the student through an existing computational package like BMD or SPSS. The need can only be resolved by an evaluation of the capabilities in the context of the subject matter for which the courseware is being developed. Our recommendation would be for APL, however, teachers in the grade schools would likely want Basic as that is likely one of the only languages with which they have had extensive use because of its availability on micro computers. For some courses in the sciences and engineering at the post-secondary level, it is likely that Fortran would be useful because of the excellent numerical functions and subroutines available in this programming language, e.g., IMSL.

The amount of power the instructor wishes to have accessible by the student is a function of the subject matter and the instructor, particularly the instructor's objectives. For example, some instructors "teach" statistics by having students run canned statistical programs such as SPSS, others would not view this as adequate, and would rather have the student see probability for example, in the context of the form and shapes of such distributions as the normal curve, Poisson distribution, the Gamma and Beta distributions, etc.

It is useful in mathematics instruction, and in some of the sciences, to have within the courseware interpreter or compiler an arithmetic function evaluator. This evaluator can be used both for answer judging and for numerical computation by the author and student. This evaluator should accommodate the evaluation of arithmetic expressions which use in addition to the basic arithmetic operations, the trigonometric, logarithmic, and exponential functions.

14.5.17 Comment Option

The capability of the author being able to insert comments within the courseware is an important support feature. This information can also be used to build a course map of the content. In addition, it is also very important for purposes of courseware maintenance. Where the courseware is stored in special data structures which are not accessible by general editors, then special software must be provided to access the comments. In this implementation comments could be inserted into the courseware by the author from the main menu and from the text menu. The comment had a particular form so that the author could search for all comments using a regular visual editor.

15.5.17.1 Suggestions and Recommendations

At least one type of comment should be allowed the author. If this comment has a specific format it can then be searched for and presented to the author in the context of the courseware created. Consideration might also be given to more than one type of comment. For example comments describing the content of the course at a particular point in the courseware might have a different identifier than a comment referring to a particular method of answer analysis being employed. The first type of comment might be used to form a course map, or presented during a "browse" mode of the courseware.

15.5.18 Branching Option

Every computer program has to have the capability of branching conditionally, or unconditionally. How this is made available to the author is a major problem. Modern computing science programming concepts argue against the use of explicit branching techniques such as the "goto" statement. The use of such statements can make a program almost incomprehensible and almost impossible to determine whether it has any bugs.

If the internal representation of the courseware is linear, as it was in our first implementation, there is no way of getting around the problem of using branching statements. It was in part for this reason, that the design of a second implementation was made and started. Although within the first implementation of the authoring system the branch was provided to the author, it did not create any serious problems because of the nature of the courseware created. However, our past experience suggests that some authors particularly when given access to the authoring language, have developed courseware almost unserviceable because of the complexity of the branching code used.

14.5.18.1 Suggestions and Recommendations

Although the internal representation of a course may make use of explicit statements of conditional and unconditionally branching, if possible the author should not be allowed to create such statements. The concepts involved in structured programming should be the guide in this instance.

14.5.19 The Run Option

The run option was provided in order that the author could:

1) observe the course as a student would see it
2) identify errors or bugs
3) to provide continuity in course development prior to making an addition or insertion of new content
4) test the adequacy of peripheral processes

The run option in the first implementation had to be tailored to the requirements of the internal courseware representation. This presented a number of problems as the courseware was in strictly linear form. The main problem which had to be anticipated was that the run option would be invoked at a point in the courseware where some parameters were not defined. For example, if the arithmetic statement A=B/C was encountered but the invocation of the course was initiated before C had been defined, then a division by zero could occur. Since this type of problem cannot be identified prior to running the course, the only method of handling it is to suspend execution and ask the author to define the variable.

There is a need for the author to be able to simply execute the courseware created. Authors will frequently get error reports from instructors and students and require a method of first identifying whether the report is accurate.

Execution of the courseware in a debug mode was not fully implemented as part of the authoring system in our first implementation because it already existed as an optional break point in the interpreter i.e., the interpreter action could be stopped and specific information retrieved.

The need for the author to be able to execute the course from a given point just prior to where new material would be created, was considered important in order for the author to continue with the addition to the last screen display, and for purposes of providing the author with continuity in the subject matter.

14.5.19.1 Suggestions and Recommendations

The major problems encountered in defining the functions of the run option were all related to handling in a transparent manner the files containing the courseware representation. For example, entering the authoring system for running a specific chapter required the opening of a file in read mode; the author then might wish to insert some new material within this chapter and this would require closing the file, copying it to the point where the insertion was to be made, opening the file for writing, adding the new material, and then appending any remaining portion. In computing software, there is elegance in simplicity, and the techniques used could not be considered simple. A better approach would be to maintain a special data structures which would be used to keep track of all file I/O status, or better to maintain the course representation in a list structure with forward and backward pointers. The latter structure could then also be used as a basis of developing an effective editor.

14.5.20 The Edit Option

The edit option is another good example where the capabilities within the authoring system become intimately dependent upon the structure of the internal representation of the courseware. Ideally, an edit option should provide edit capabilities for all aspects of the courseware created by the authoring system, first during the creation phase, and secondly after the courseware has been created. Ideally, the editing should be directly on what the author sees on the screen as well as those features of the courseware which do not produce screen effects. Editing must be available for graphics as well as text, and if special technical capabilities are available, such as audio, then editing of audio may be required. Satisfying this range of needs is no minor task, and more research is needed.

The editing capabilities of our first implementation allowed editing of text as it was being placed on the screen by the author back to the point where an escape key had been used. This type of editing is a function of the operating system under which the authoring system is created. A second pass at editing was provided by executing the courseware and making changes as they were identified. (A third pass at editing could be made by the author by using any one of the editors available on the computing system on which the implementation was made).

The invocation of the editor required the author to identify the beginning location at which editing was to begin. This execution was limited to those elements of the courseware which would produce results on the screen, and not those elements which did not cause an effect on the screen. Both types of elements were however displayed for the author at the bottom of the screen in the form of their representation within the courseware. The author would then be presented with the options to continue, to edit, replace, delete, or insert an element. If an edit was selected, the statement to be edited was written in a cyan color to indicate edit mode, and the line could be edited directly by appropriate positioning of the cursor.

The internal representation of the courseware was placed into a list structure with forward pointers beginning at the point in the course defined by the author for up to a maximum of 150 elements. At the end of parsing through these elements the process was restarted automatically at the beginning so that the author could make further checks.

14.15.20.1 Suggestions and Recommendations

The following factor should be considered in designing software for editing courseware by the author:

1) The nature of the elements within the courseware must be identified and a decision made as to what editing capabilities will be provided on these elements.

2) The nature of the internal data representation holding the courseware becomes a crucial factor in the editing capabilities which can be provided. The appropriate

data structure appears to be that each element to be subject to editing should be held in a list structure with backward and forward pointers so that complete flexibility exists in inserting, deleting, cutting and pasting. Consideration also has to be given to the length of the list structure relative to the amount of memory which is available for holding the elements to be edited. Our experience suggests that a single chapter may have as many as 8000 elements which are subject to editing. Of course, this length can be restricted to a smaller number during the authoring process. If the internal representation of a course is not in a list structure, then it will have to be placed into such a structure for the editing function.

3) Editing should be performed on observable screen effects when such effects are present. This means a synchronous relationship between the cursor position on the screen and the internal representation. Usually, this is not a problem, but consideration must be given to the attributes of the elements to be edited, e.g., color changes, font changes, superscripts, subscripts, etc. In other words, even though the author is editing elements which have observable screen effects, there are components which are not observable as discrete elements on the screen.

4) Ideally, a single editor should handle graphics and text editing. This implies that the editor must know the data structures it is working on as directed by the position of the cursor on the screen. This is no minor task, and should be the subject of more research.

14.15.21 The Exit Option

The exit option in our implementation initiated the housekeeping functions which were required to maintain authoring information for the next session. A number of files where created transparently for the author:

1) A new version of a course file was created for the chapter which was worked on. The old version would normally be retained for backup purposes. This feature would likely vary with different operating systems.

2) Continuity files where created for each chapter, and one for the whole course. A continuity file for a chapter included information such as the labels which were created and associated comments. The continuity file for the whole course would contain, for example, a list of the graphics and their associated notes.

All these files could be examined directly using any one of the editors provided by the operating system.

14.15.21.1 Suggestions and Recommendations

It would be useful for research purposes to maintain a login file holding information about the author's authoring characteristics. These characteristics might include the following:

1) A time/date of the logging on, off, and time in authoring mode.

2) The number of times each authoring system option was used, categorized by the functional areas of the authoring system.

3) The amount of courseware generated by the authoring system.

4) The amount of time spent in the major functional components of the authoring system.

5) The number and location of error messages provided to the author.

14.2 Generality of Authoring Systems

During the last couple of years numerous authoring systems have appeared. Most of these appear to be distributed by commercial firms. These firms include the traditional computer manufacturers, software development companies, and private entrepreneurs. Advocates of each systems extol virtues of each system, sometimes on the basis of a cursory use of the system and sometimes by reference to documentation, but seldomly based on extensive use in a non-trivial environment. Some advocates are sufficiently bold to even offer workshops on how to use a particular authoring system which they have no experience in the creation of any substantial amount of courseware!

The choice of which system to use depends upon a number of very practical considerations such as cost, availability of equipment on which the system runs, experience with similar systems in the past, ease of use, reliability, developer support services, and the nature of the instructional needs.

Our research suggests that no single system will satisfy a wide range of authoring and courseware design needs unless that system has components which are sufficiently modular that they can be selected for inclusion in the system on the basis of the authors needs. This conclusion is not surprising if one examines the curricular content of any single grade level, a year of technical or university instruction in one specific program, or the training needs of a specific commercial firm.

The modularity which we suggest is illustrated in Figure 14.13.

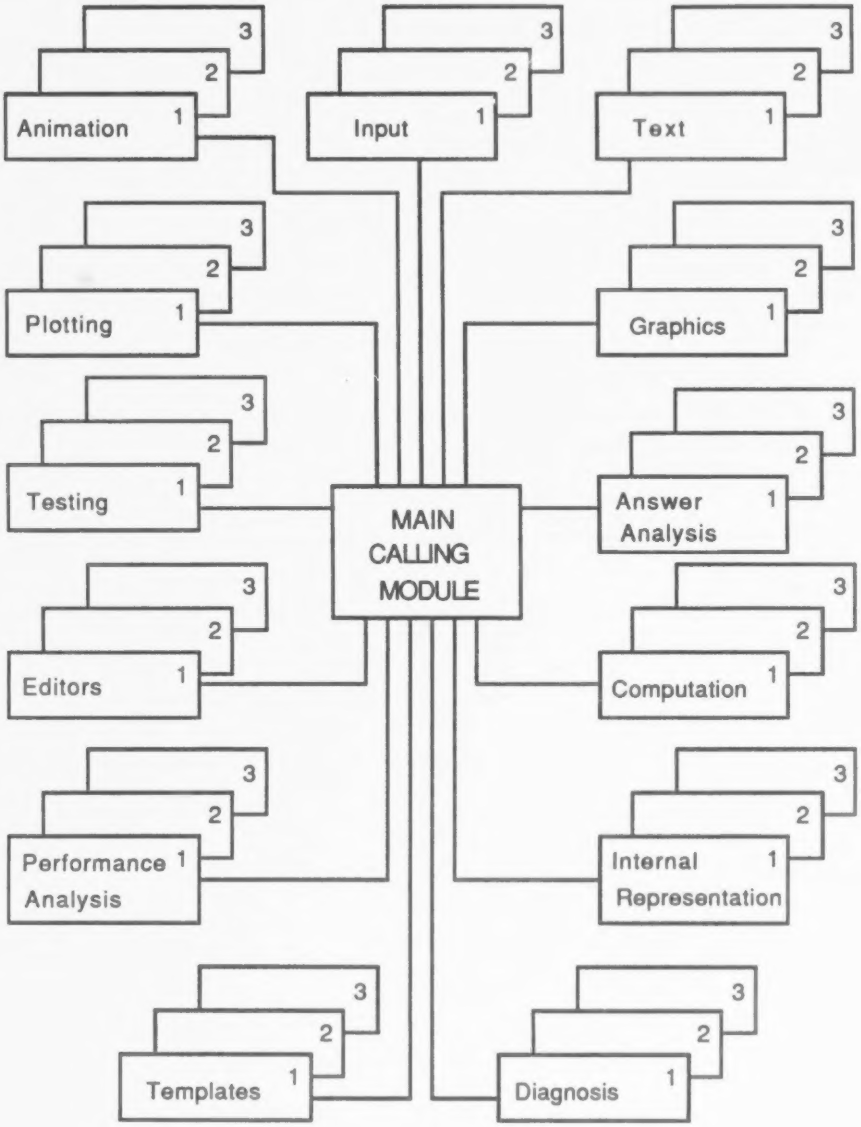

Animation
Input
Text
Plotting
Graphics
Testing
Answer Analysis
MAIN CALLING MODULE
Editors
Computation
Performance Analysis
Internal Representation
Templates
Diagnosis
1
2
3

Figure 14.13 Modular Components for Selection To Meet Authoring Needs

With a highly modular system, a user could then select those components required for the task at hand. For example, the graphics capability selected to create a course in electronics would have different capabilities than that required for physics or chemistry. The software developer of the system should be able to add new components or replace old components as user needs arise.

Chapter 15
Second Implementation

15.0 Introduction

This second implementation of an authoring system was concerned with the definition of an internal representation of course structure, to alleviate the problems inherent in a strictly linear representation as used in the first implementation. The first implementation revealed a number of problems which related to an inadequate definition of the edit space and the point in the course at which execution could be initiated by the author. In addition, inherent to the strictly linear representation of the courseware were the problems related to the effective and easy maintenance of files, and problems of courseware reliability and maintenance. The second implementation also was designed to be highly modular in the functional components used, so that these components would be open to easy change, or substitution depending upon the needs of the author. The second implementation did not concern itself directly with the author interface to the system.

15.1 General Hierarchical Structure

A hierarchical tree structure was designed for structuring the courseware in the second implementation. The hierarchical structure also provided the basis for defining the scope convention (the domain over which some entity would be known) for a number of functional sub-components of the system including variables. The hierarchical structure as it was applied to the organization of the course is illustrated in Figure 15.1, although the naming of the nodes in this figure are arbitrary, i.e., the names can be defined by the author, and are included for illustrative purposes only.

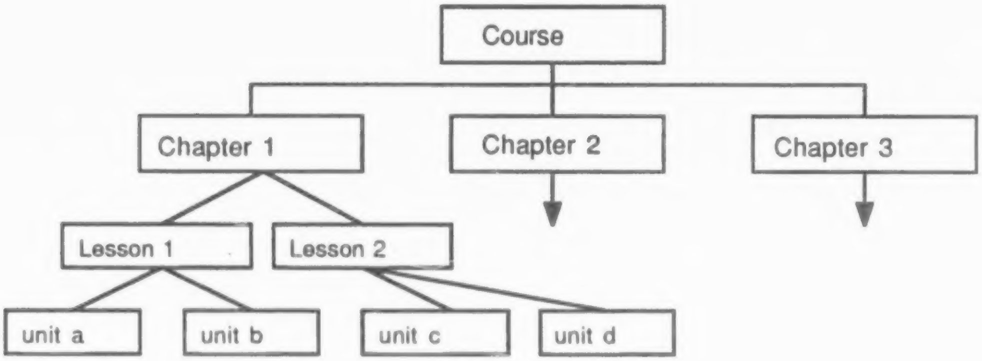


Figure 15.1 Hierarchical Course Structure

An author could, for example, define each level in the tree structure with names other than "course", "chapter", "lesson", and "unit".

Each node point has its own control module and associated records. The control module at level n was designed to have the following controlling capabilities:

1) invocation of a content module at levels n-j, where j could be 1, 2, 3, ... steps higher in the hierarchy

2) invoke a content module at level n

3) reference routines, functions and variables at levels n-j

4) reference routines, functions and variables at level n

A content module would be the basic unit within which the course content and instructional functions are resident.

The controlling functions of the control module are illustrated in Figure 15.2

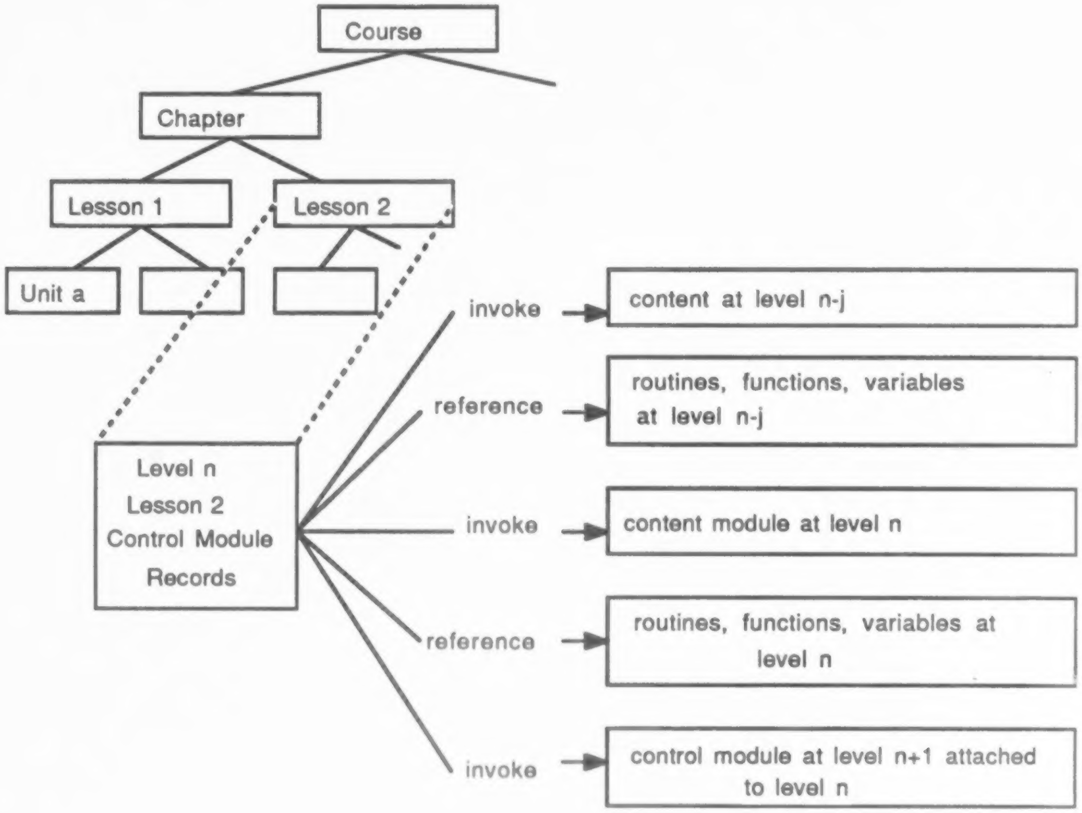

Course
Chapter
Lesson 1
Lesson 2
Unit a
Level n
Lesson 2
Control Module
Records
invoke
content at level n-j
reference
routines, functions, variables at level n-j
invoke
content module at level n
reference
routines, functions, variables at level n
invoke
control module at level n+1 attached to level n

Figure 15.2 Functional Scope Space At Level n

The content module was designed to provide the access to the basic instructional functions of the courseware, e.g., display to the screen, input, answer analysis, etc.

In order to provide for flexibility in substituting and/or modifying existing components of the authoring system, as well as for defining a meaningful set of expressions related to specialized functions, an interpreter capable of interpreting the expressions in each functional unit was developed. Interpreters were developed for each of the following:

1) control language
2) content language
3) display language
4) input language
5) answer language
6) menu language

It is the function of each of the interpreters to understand and execute the specific expressions which form part of the internal representation of the courseware. For example, the input language interpreter would be able to interpret the expressions in the internal representation of the courseware which would be required to obtain an answer from the student. The development of separate interpreters was facilitated by the Elf language developed in the Division. The syntax of these languages are given in Appendix C.

15.2 The Control Language

The control language is used to define the internal representation of control modules which control the student movement through the content modules available within the scope of a current control module, and the control module of the next level. This language is a structured procedure oriented language which would be created through an author interface to the authoring system, i.e., most authors would not be aware of its existence or its form.

The control language has the following capabilities:

1) the definition of control, routine, and function processes, the latter return string or numeric results.

2) arithmetic and logical expressions

3) the definition of numeric and string variables and reference to strings, lists, and record types

4) the semantics include looping, while, until, for, if, and case structures.

5) calls may be made to system routines and to an Elf grammar

5) assignment operations include numeric and string assignment including numeric increment and string append, as well as conversion of variable types.

The syntax of the control language is given in Appendix C.

15.3 The Content Module and Language

The content module provides the internal representation of content for presentation of to the student, the processing of the student's responses, including answer analysis and reprise statements. The content module may call upon other modules as illustrated in Figure 15.3 shown below.

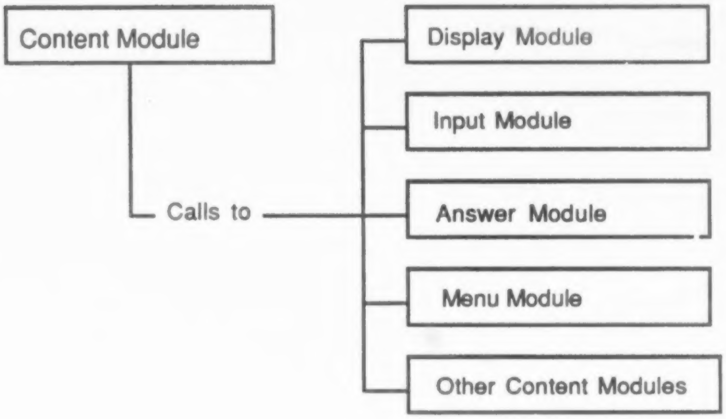


Figure 15.3 Calling Domain of the Content Module

The content module also has provisions for the definition of constants, variables, and expressions similar to the control module, and calls to routines. The content module may also contain statements which cause text and graphics to be placed on the screen. The syntax of the content language is given in Appendix C.

15.4 Display Language

The display language is used to specify the presentation of text and graphics to the screen. The display language statements are interpreted by the display language interpreter, for which the syntax is given in Appendix C.

The general form of ~~the~~ display statement body is similar to that used in the content statement body, however, provision is made for specifying the processes for display of text and graphics to the screen. The latter processes include the following:

1) textbox: provision for specifying the window in which text will appear

2) drawbox: provision for specifying the window in which a graphic will appear

3) erase: erases a named textbox or drawbox

4) wait: allows specification of the time for a pause in execution which may be until any key is pressed

5) text: specification of the text to appear in a textbox

6) deletebox: deletion of a box

7) draw: specification of the moves required to draw an object within the drawbox

The textbox and drawbox are the basic screen units within which text and drawings can be presented to the student. The two units may be positioned anywhere on the current screen. The textbox has a internal coordinate system having an origin in the upper left corner, i.e., (1,1) for row 1, column 1, while the drawbox has a traditional x-y coordinate system with an origin (0,0) located anywhere. The coordinate system used in the boxes is a variable open to default or specific definition by the author. Text and draw boxes may have an identifying name associated with them, and may have the attributes of being "framed", "local", or "translucent". Boxes may be "erased" and "deleted".

Text is placed in a textbox based upon the internal coordinate system of the box. Thus, text of varying sizes can be accommodated. Associated with a textbox is the optional definition of a message to the student indicating how to control the rate of presentation, e.g., a message to touch the space bar to continue. Text within a textbox can be accumulated by successive writes to the same box with automatic positioning at the next available row. Text within a box may be subject to block attributes such as assignment of left and right margins, indents, tab use, separation between adjacent lines, text color, centering, and holding paragraphs as complete units. Text may also be displayed in an "as is" mode, or formatted on output to the display box.

In its initial specification, it should be noted that the textbox becomes the basis for displaying text on terminals with characteristics different than that on which the author created the text. The textbox feature is also supported by a screen support system written in Elf, and which has been used to demonstrate portability of Tutor code running on a DEC VAX system with output to a DEC Gigi terminal, a MacIntosh, and an IBM pc.

The flexibility associated with the textbox is also characteristic of the drawbox used for the display of graphics to the student. Upon termination of this project a full graphics facility had not been implemented for the drawbox, however, a simple drawing feature was implemented.

15.5 The Input Language

The Input language is a general facility which allows a response to be taken from the student. Two modes of input were implemented, one being the input of one or more choices from a list, while the other a string input. The facility is general in the sense that automatic formatting of the choices, and movement of the cursor to the position of the choices is inherent to the software.

The input language has the same general form as the other languages noted earlier. In addition for both modes of input the basic area of the screen within which the input is occurs is the "box" or window.

The definition of input in the choice or selection mode can have automatic positioning of the cursor beside the choices within the box, a definition of the usable keys, the manner of identifying the choices, e.g., digits, or characters in lowercase or uppercase, the maximum wait time, and the number of valid selections which must be made. The generality of the input language was planned so that it could be used not only as part of the definition of courseware, but also as a part of the authoring system software.

The definition of input from the keyboard was also contained within the domain of a box. The starting position of the cursor could be defined, and text is formatted within the boundaries of the box.

15.6 The Answer Language

The answer language was defined for the specification of the processes required to check or judge a student's answer. As with other languages developed, the general format of the body of the internal representation of the answer language had a common form. The answer language, in addition to specifying the procedures for answer analysis, provided access to the display of a reprise statement, and a retry option. A more detailed description is given in Appendix C.

The following system-type routines, written in Elf, were developed for cleaning up input and determining its adequacy. The "cleaning-up" routines were as follows:

1) character delete: deletes from the input all characters defined in a key string returning the number of deletions and their positions in a mask

2) character delete extra: deletes duplicate characters returning the number of deletions and their positions in a mask

3) character purge: as in (2) but does the delete at the beginning and end of the input string

4) character search: searches the input string for key characters; returns number found and their position

5) character substitute: substitutes a defined character for another in the input string

6) substitute: string "a" is replaced by string "b"; the number of occurrences is returned

The general software building blocks upon which the answer judging is build are listed below, and appear in more detail in Appendix C under Answer Language.

1) scan: searches from position p to p+j for a given string; returns information defining where string was found

2) pattern scan: similar to scan above, but the key may contain special default or "wild card" character and string definitions

3) similar: a measure of similarity between a key and the input string

4) spell: uses the edit distance match for checking for spelling errors; information is returned which can be used in a markup procedure

5) num: used for checking the range of a numeric value; start position and end position for the value can be specified

6) nums: as in (5) above, but capable of handling vectors

15.7 Menu Language

The menu language was designed more specifically for use in creating the author interface to the authoring system. It is similar to the input language described under Section 15.5, but has greater flexibility in specification of statements which appear inside the box displaying the choices.

Chapter 16

Future Research Considerations

16.0 Introduction

Research in any discipline cannot be characterized as moving along uniform wave fronts in the manner analogous to the waves created by dropping a pebble into a pond of water. There is research which is required in the development of authoring systems to simply increase the efficiency of producing courseware. There is also the need for research into the more complex task of making the systems under which instruction is offered more adaptable to the needs of the learner. There is a need for delivery systems to be more intelligent. The rate of progress in each of these areas will not be uniform.

What is possible, and perhaps desirable, from a computing science point of view is not necessarily what authors perceive to be desirable or important to ameliorating the problems related to the development of courseware. Authors may frequently underestimate the computing complexity inherent in their requests for certain authoring facilities.

16.1 Areas in Need of Study and Research

1) **Author needs:** There is likely a common set of processes and functions which are required regardless of the nature of the courseware being created. However, there seems to be little research on what instructors or teachers at different levels of our educational system actually desire. It seems unreasonable to assume that the needs of an author creating courseware for use in elementary school are the same as those of an author creating courseware for use in the armed forces; the authoring needs in the area of foreign language instruction are likely to not be the same as those required in the area of chemistry at the university level. The problem is compounded because so few instructors seem to know what CAI could accommodate from an instructional point of view, and some are still strongly opposed to the entry of computers into their instructional environment (Hunka, 1987). Because some instructors are so unfamiliar with CAI, the validity of opinions in this matter is likely to be suspect. However, instructors and teachers should be able to define the instructional tasks, as well as broad areas of instructional concern which may be accommodated by CAI approaches.

2) **Authoring styles:** It is unlikely that all authors have the same "style" of authoring, or are willing to accept a "team" approach. For example, two CAI based credit courses at the University of Alberta which have been in operation for well over ten years were not constructed using a team approach. They were designed by a small number of content specialists. Some authors are likely not to be willing to explicitly use a "systems approach". There is likely to be more than one authoring style which can be used to produce effective courseware. Authoring systems should be able to accommodate a number of different styles.

3) **Authoring Models:** Some authoring systems, such as Course of Action require the author to first represent a sequence of instruction using a fixed set of icons, and then to create the screen displays. Another approach is for the author to first create what he wants the student to see, and have the system build the course structure. Using an authoring language requires the author to be even more

distant from what is actually presented to the student. How well do these models work with different instructors creating different kinds of courseware in different subject matter areas?

4) **Understanding the Authoring System:** Experience with software tends to build in the user a conception of the software structures, idiosyncrasies, and procedures which do and do not work. Sometimes these conceptions accurately reflect the nature of the software, and sometimes they do not. How easy is it for an author to create an accurate conceptual structure of an authoring system? What support can be given the author in order to make it easier to develop an appropriate cognitive representation of the authoring system?

4) **Answer Judging and Diagnosis:** The adequacy of identifying the nature of the student's response represents the intelligence of a CAI system to the user. It is likely in this domain that AI techniques can be applied with considerable benefit. It is also likely that unique algorithms are required in different subject matter areas. Consideration must also be given to the effects of "noise" in the student's input and how it may be removed. The analysis of graphical input by the student still requires considerable work in defining the basic algorithms required. To improve upon the usual categories correct, wrong, unrecognized, a reply to a student warrants in many cases a specific reason why an answer is not completely acceptable. Algorithms which may have common elementary operations, but which can be combined to provide diagnostic capabilities in different subject areas are also required. Expert systems and AI techniques would likely be useful here.

5) **Representation of Transparent Processes:** Some processes during authoring do not create an observable screen effect. Some of these process, such answer judging can be complex and require a representation which can be meaningful to the author.

6) **Student Models:** There appears to be different conceptions of what is a student model, the nature of its contents, and its purpose within the run time CAI system.

7) **Benchmarking and Measuring Efficiencies:** Because many authoring systems are commercially produced, their descriptions tend to be designed to promote sales. Benchmarking and procedures for measuring efficiencies are required in addition to functional characteristics. Benchmarking should be related to author efficiencies, usefulness in specific subject matter areas and instructional needs.

8) **General Editors:** Editors which can handle textual as well as graphical representations in a consistent manner are required.

9) **Integrated Systems:** Components of an authoring system require subsystems which can be integrated transparently for the author. Although different programming languages might be used to produce such subsystems, these subsystems must be accessible from within the authoring system.

10) **Graded Developmental Steps:** If useful authoring systems are to be created, design characteristics should include the possibility of incremental additions (and deletions) of subsystem components. The design should include consideration of the run-time terminals used by students being different in different

instructional environments, as well as the use of more technologically advanced terminal.

11) **Debug Environment:** A subsystem which facilitates the location of errors in the courseware is required as part of the authoring system. This subsystem should maintain as close as is possible the run-time environment, but provide the author with capabilities of efficiently isolating and correcting errors. All errors may not be in the courseware, as some errors may reside in peripheral systems such as those creating performance records.

12) **Curricular Organization Support:** A subsystem is required to assist the author with maintaining integrity of the curricular organization, not only as the courseware is being created, but after several modifications.

Bibliography

Allen, B.S., Merrill, M.D., System-Assigned Strategies and DBI. Jr. Educational Computing Research, Vol 1(1), 1985.

Authorware Inc., Reference Manual: Course of Action, Authorware Inc:, Minneapolis, Minn, 1987

Avner, A., Smith, S., Tenczar, P., CBI Authoring Tools: Effects on Productivity and Quality, Jr. Computer Based Instruction, Vol 11, No.3, 1984.

Avner, R.A., and Tenczar, P., The Tutor Manual. CERL Report X-4, University of Illinois, Urbana, January 1969.

Bagnall, K.M., Szabo, M., Hall, S., and Jensen, W., The Evolution of an Authoring System on Plato, Jr. of Computer Based Instruction, Vol 11, No. 3, 1984.

Bourneque, R., Treu, S., Specifications and Generation of Variable Personalized Graphical Interfaces, Int. Jr. Man-Machine Studies, 22 ,1985

Brahan, J., and Godfrey, D., The Natal Language, Press Procepic Ltd., Victoria, B.C., 1984.

Brockenbrough, S.A., and Merrill, M.D., System Assigned Strategies and CBI, Jr. Educational Computing Research, Vol 1(1), 1985.

Brown, P.G., Herot, C.F., and Souza, P. Program Visualization: Graphical Support for Software Development, Computer, IEEE, August 1985.

Brown, P.J., Error Messages: The Neglected Area of the Man-Machine Interface? Communications of the ACM, April, 1983.

Card, S.K., Moran, T.P., and Newell, A. The Psychology of Human-Computer Interaction. Lawrence Erlbaum Associates, Publishers, Hillsdale, N.M., 1983.

Chang, S., Visual Languages: A Tutorial and Survey, Software, IEEE, January, 1987.

Clancey, W.J. Qualitative Student Models, Annual Review of Computing Science, 1986.

Control Data Corporation, Plato Author Language Reference Manual, Control Data Corporation, St. Paul, Minn., 1978

Davis, A., Elf Reference Manual, Version 5 Level D, Division of Educational Research Services, University of Alberta, Edmonton, 1985.

Digital Equipment Corporation, Dimension Authoring System, User's Guide, AA-K764A-TE, Maynard, Mass., July 1982.

Dowsey, M.W., Easy Author-entry Systems: A Review and a Prototype, International Jr. Man-Machine Studies, 1974.

Efe, K., A Proposed Solution to the Problem of Levels in Error Message Generation, Communications of the ACM, Vol. 30, No. 11, November 1987.

Elithorn, A., and Banerji, R. (Editors) Artificial and Human Intelligence, Papers presented at the International NATO Symposium on Artificial and Human Intelligence, Lyon France, October 1981, North-Holland, Amsterdam, 1984.

Fairweather, P.G., and O'Neal, A.F., The Impact of Advanced Authoring Systems on CAI Productivity, Jr. Computer Based Instruction, Vol. 11, No. 3, 1984.

Farley, B., Henneker, W.H., Hlady, A.M., Orchard, R.A., and Phan, C.S., MicroNatal: A Portable Approach to CAL Courseware Creation and Delivery, National Research Council, Ottawa, 1986.

Foley, J.D., Wailace, V.L., and Chan, P., The Human Factors of Computer Graphics Interaction Techniques, Computer Graphics and Applications, IEEE, November, 1984.

Gaines, B.R., and Shaw, M.L.G., From Timesharing to the Sixth Generation: The Development of Human-Computer Interaction Part I. International Jr. of Man-Machine Studies, Vol 24, January, 1984.

Gait, J., Pretty Pane Tilings of Pretty Windows, Software, IEEE, Vol 3, No. 5, September 1986.

Heck, W.C., and Grimm, P.E., Ease: Easy Authoring System for Educators, Jr. Computer Based Instruction, Vol. 11, No. 3, Summer, 1984.

Hillelsohn, M.J., Benchmarking Authoring Systems, Jr. Computer Based Instruction, Vol. 11, No. 3, Summer, 1984.

Honeywell Information Systems, Natal-II Language Specification Manual, Toronto Software Development Centre, Toronto, 1981.

Houghton, R.C., Online Help Systems: A Conspectus, Communication of the ACM, February, 1984.

Hunka, S., CAL Authoring System, Fifth Canadian Symposium on Instructional Technology, National Research Council, Ottawa, 1986.

Hunka, S., A survey of CAI Capabilities of AVCs, Colleges, and Technical Institutes, Division of Educational Research Services, University of Alberta, Edmonton, 1987.

International Business Machines, IBM 1500 Coursewriter II Author's Guide, Programming Publications, San Jose, Calif., 1970.

Jones, M.A. and Tuggle, F.D., Inducing Explanations for Errors in CAI, International Journal of Man-Machine Studies, Vol. 11, 1979.

Kearsley, G., Authoring Systems in Computer Based Education, Communication of the ACM, 25, July, 1982.

Kearsley, G., Authoring Tools: An Introduction, Jr. of Computer Based Instruction, Vol. 11, No. 3, 1984.

Kearsley, G., Microcomputer Software: Design and Development Principles, Jr. Educational Computing Research, Vol. 1(2), 1985.

Kearsley, G., Embedded Training: The New Look of Computer-Based Instruction, Machine Mediated Learning, Vol. 1, No. 3, 1985.

Kaiser, G.E., and Garlan, D., Melding Software Systems from Reusable Building Blocks, Software, IEEE, July, 1987.

Klass, R., The TenCore Language and Authoring System for the IBM Personal Computer, Jr. of Computer Based Instruction, Vol. 11, No. 3, 1984.

Lee, D., Graphics Input and Analysis for Computer-Assisted Instruction, M.Ed. dissertation, University of Alberta, Fall 1983

Long, K.T., Squire, E.C., Courseware's Apple Authoring System. Jr. of Computer Based Instruction, Vol. 11, No. 3, 1984.

McCann, P.H., Methods for Improving the User-Computer Interface. Navy Personnel Research and Development Center, San Diego, Calif. NPRDC TR 83-29, 1983.

Martin, M.P., and Fuerst, W.L., Using Computer Knowledge in the Design of Interactive Systems, International Jr. Man-Machine Studies, 1987.

Merrill, M.D., Where is the Authoring in Authoring Systems?, Jr. Computer Based Instruction, Vol. 12, No. 4, 1985.

Merrill, M.D., Prescriptions for an Authoring System, Jr. Computer Based Instruction, Vol. 14, No. 1, 1987.

Morland, D.V., Human Factors Guidelines for Terminal Interface Design, Communications of the ACM, 26,7, 1983.

Mudrick, D., and Stone, D., An Adaptive Authoring System for Computer-Based Instruction, Jr. Computer Based Instruction, Vol. 11, No. 3, 1984.

Murch, G.M., Physiological Principles for the Effective Use of Color, Computer Graphics and Applications, IEEE, 1984.

Nesbit, J.C., Approximate String Matching in Response Analysis, Jr. of Computer-Based Instruction, Vol. 12, No. 3, Summer 1985.

Nesbit, J.C., Applications of Learning Hierarchies in Adaptive Instructional Systems, Ph.D. dissertation, University of Alberta, Spring, 1988.

Norman, D.A., Design Rules Based on Analyses of Human Error, Communication of the ACM, 26, 7, 1983.

O'Shea, T., Bornat, B., Cu Boulay, B., and Eisenstadt, M., Tools for Creating Intelligent Computer Tutors, in Artificial and Human Intelligence edited by Elithorn and Banerji, North-Holland, Amsterdam, 1984.

Osin, L., SMITH: How to Produce CAI Courses Without Programming, International Jr. of Man-Machine Studies, Vol. 8, 1976.

Reed, M.J., and Porter, M., Review of Digital's Courseware Authoring System, Jr. of Computer Based Instruction, Vol. 11, No. 3, 1984.

Reilly, S., and Roach, J.W., Designing Human-Computer Interfaces: A Comparison of Human Factors and Graphic Arts Principles, Educational Technology, 1986.

Romaniuk, E.W., and Schienbein, R., CWII Function Guide to the DERS CAI System, Division of Educational Research Services, University of Alberta, 1973.

Schneidewind, N.F., The State of Software Maintenance, IEEE Transactions on Software Engineering, Vol. SE-13, No. 3, 1987.

Self, J.A., Student Models in Computer-aided Instruction, International Jr. of Man-Machine Studies, Vol. 6, 1974.

Sherwood, B.A., The Tutor Language, Control Data Education Company, Control Data Corporation, Minneapolis, Minn., 1977.

Schultz, D., Omnisim: User's Guide Version 1.2, Instructional Systems Group, Computing Services, University of Alberta, Edmonton, 1986.

Snowberry, K., Parkinson, S., and Sisson, N., Effects of Help Fields on Navigating Hierarchical Menu Structures, International Jr. of Man-Machine Studies, April, 1985.

Tatsuoka, K.K., Toward an Integration of Item-Response Theory and Cognitive Error Diagnoses, CERL Technical Report 87-1-ONR, University of Illinois, July 1987.

Wainer, H., and Thissen, D., Plotting in the Modern World, Technical Report No. 86-70, ETS, Princeton, N.J., 1986.

Weiser, M., Source Code, Computer, IEEE, Vol. 20, No. 11, 1987.

Westrom, M., National Author Language: Specification Manual, NRC 14245, National Research Council, Ottawa, Canada, 1974.

Wolman, R. (Editor), Survey of Authoring Systems, Data Training, Mar/May, 1986.

APPENDIX A
Photographs of Some Screen Displays Experimental Authoring System (XAS)

SSHRC PROJECT NO. 410-84-1111-R2

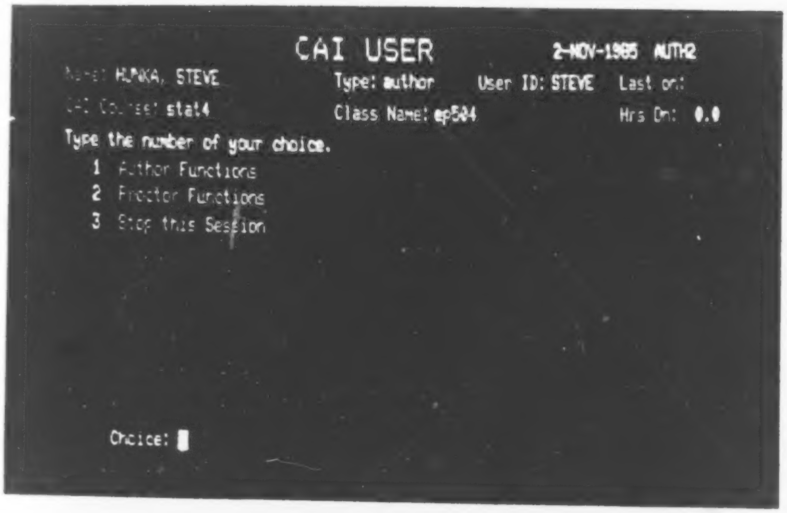
S. Hunka

Division of Educational Research Services
Faculty of Education
University of Alberta, Edmonton
February 1988

Notes to Photographs

Photograph Number	Note
1	1st menu upon entering the CAI system
2	2nd menu upon entering the CAI system
3	Display of authoring parameters and summary information of course being authored
4	Example of 45 column screen format and main menu
5	Example of 80 column screen format
6	Example of split screen format
7	Example of text placed on screen and text options
8	Example of edit mode for internal text statement
9	Text mode with display of font set
10	Example of placing box around title
11	Example with boxes around text completed
12	Effect actions in 10 and 11 as seen by student
13	Creation of drawing using cross hairs
14	Drawing with surrounding text
15	Example of drawing called from library
16	Defining a target area (d) for point response
17	Example using defined target area as seen by author
18	Defining a Boolean standard for answer judging
19	Example of a pattern recalled for Boolean analysis
20	Example of edit menu for use with pattern
21	Examples of point and vector plotting at label stage
22	Point and vector plots as seen by student
23	Example of diagram as seen by student

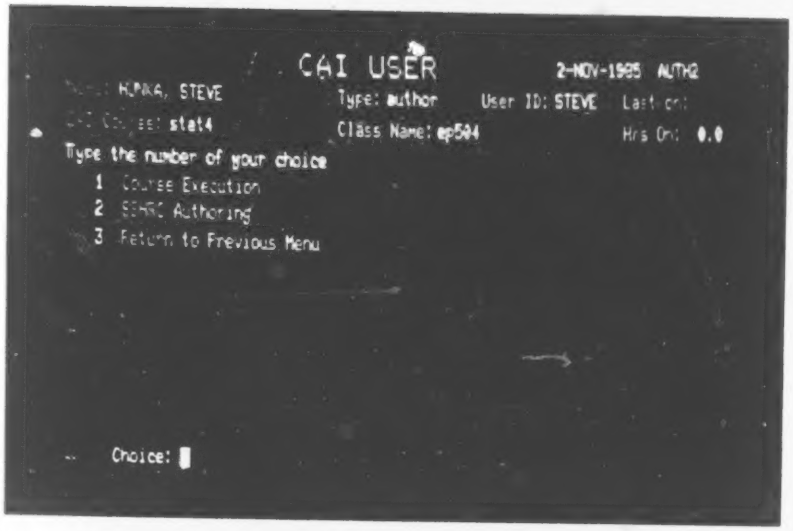
1 1st menu upon entering the CAI system



1

```
                         CAI USER            2-NOV-1985  AUTH2
Name: HUNKA, STEVE               Type: author     User ID: STEVE   Last on:
CAI Course: stat4                Class Name: ep504                 Hrs On:  0.0
Type the number of your choice.
   1  Author Functions
   2  Proctor Functions
   3  Stop this Session




        Choice: █
```

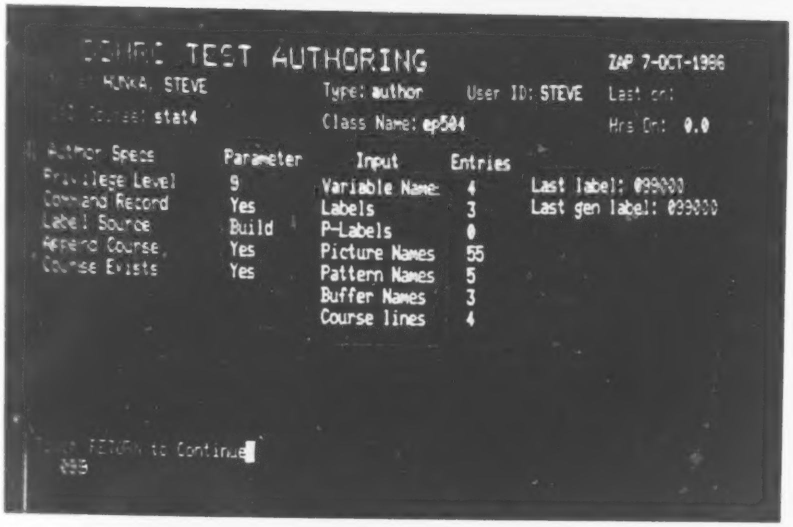
2 2nd menu upon entering the CAI system



2

```
                         CAI USER            2-NOV-1985  AUTH2
Name: HUNKA, STEVE               Type: author     User ID: STEVE   Last on:
CAI Course: stat4                Class Name: ep504                 Hrs On:  0.0
Type the number of your choice
   1  Course Execution
   2  SSHRC Authoring
   3  Return to Previous Menu




        Choice: █
```

3 Display of authoring parameters and summary information of course being authored

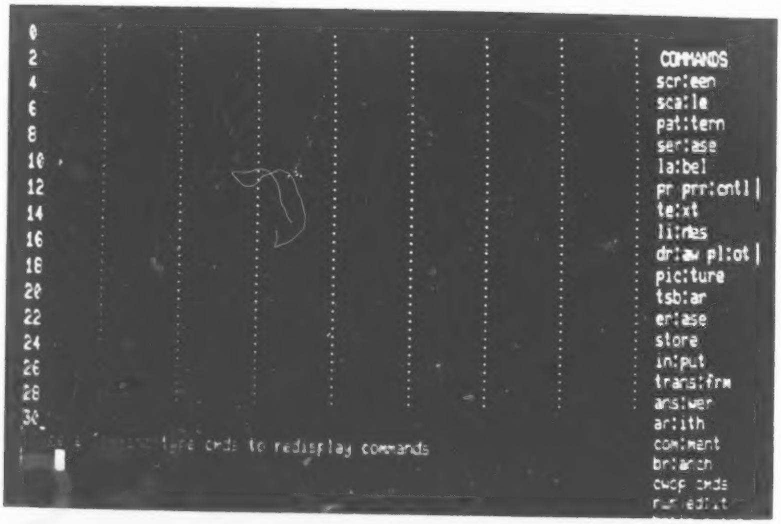


3

```
  SSHRC TEST AUTHORING                                    ZAP 7-OCT-1986
Name: HUNKA, STEVE            Type: author     User ID: STEVE   Last on:
CAI Course: stat4             Class Name: ep504                 Hrs On:  0.0
Author Specs      Parameter     Input          Entries
Privilege Level   9             Variable Name   4        Last label: 099000
Command Record    Yes           Labels          3        Last gen label: 099000
Label Source      Build         P-Labels        0
Append Course     Yes           Picture Names   55
Course Exists     Yes           Pattern Names   5
                                Buffer Names    3
                                Course lines    4



Press RETURN to Continue█
 099
```

4 Example of 45 column screen format and main menu

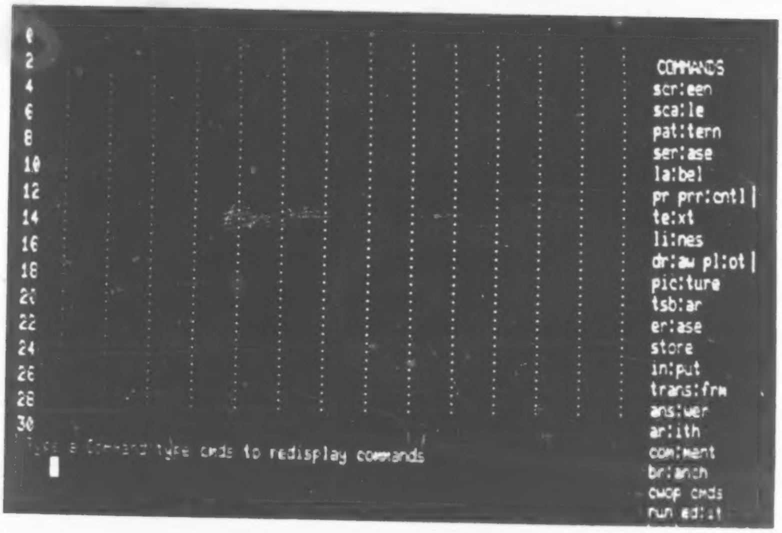
4


0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
COMMANDS
scr:een
sca:le
pat:tern
ser:ase
la:bel
pr prr:cntl |
te:xt
li:nes
dr:aw pl:ot |
pic:ture
tsb:ar
er:ase
store
in:put
trans:frm
ans:wer
ar:ith
com:ment
br:anch
cwop cmds
run ed:it

```
Type a Command/type cmds to redisplay commands
```

5 Example of 80 column screen format

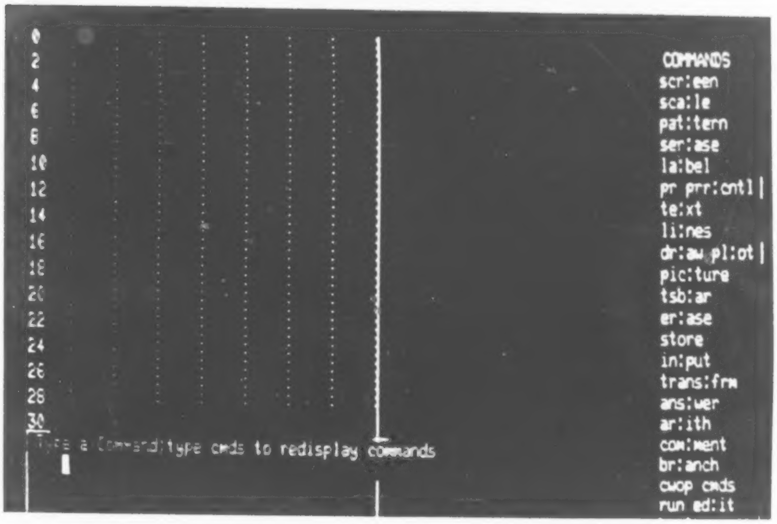
5


0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
COMMANDS
scr:een
sca:le
pat:tern
ser:ase
la:bel
pr prr:cntl |
te:xt
li:nes
dr:aw pl:ot |
pic:ture
tsb:ar
er:ase
store
in:put
trans:frm
ans:wer
ar:ith
com:ment
br:anch
cwop cmds
run ed:it

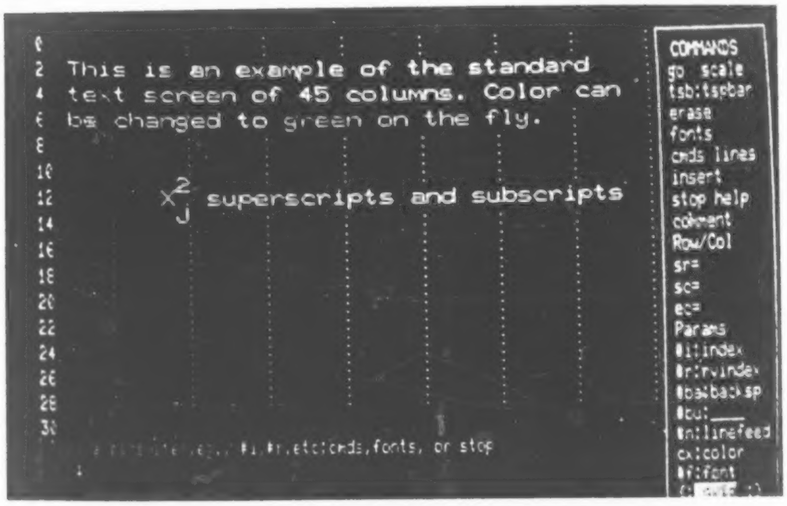
```
Type a Command/type cmds to redisplay commands
```

6 Example of split screen format

6


0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
COMMANDS
scr:een
sca:le
pat:tern
ser:ase
la:bel
pr prr:cntl |
te:xt
li:nes
dr:aw pl:ot |
pic:ture
tsb:ar
er:ase
store
in:put
trans:frm
ans:wer
ar:ith
com:ment
br:anch
cwop cmds
run ed:it

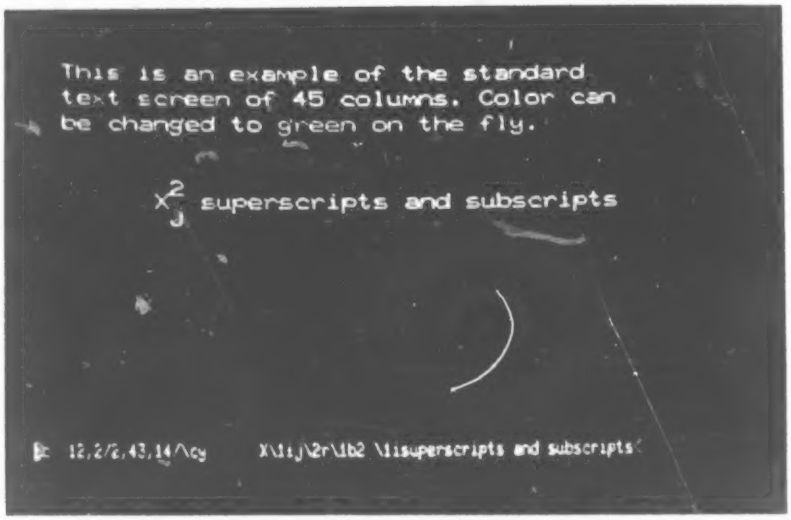
```
Type a Command/type cmds to redisplay commands
```



```
This is an example of the standard
text screen of 45 columns. Color can
be changed to green on the fly.

X^2_J superscripts and subscripts

[illegible] #i,#r,etc:cmds,fonts, or stop
```

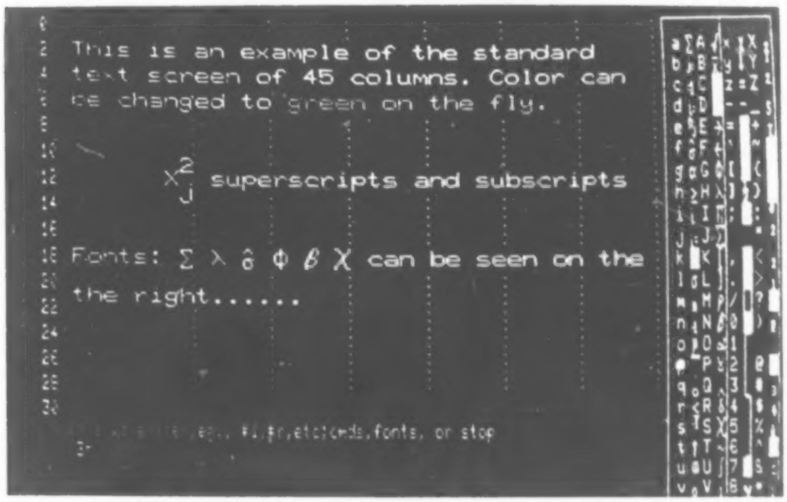
COMMANDS
go scale
tsb:tspbar
erase
fonts
cmds lines
insert
stop help
comment
Row/Col
sr=
sc=
ec=
Params
#i:index
#r:rvindex
#ba:backsp
#bu:____
#n:linefeed
cx:color
#f:font

7 Example of text placed on screen and text options



```
This is an example of the standard
text screen of 45 columns. Color can
be changed to green on the fly.

X^2_J superscripts and subscripts

c 12,2/2,43,14^cy      X\1ij\2r\1b2 \1isuperscripts and subscripts
```

8 Example of edit mode for internal text statement

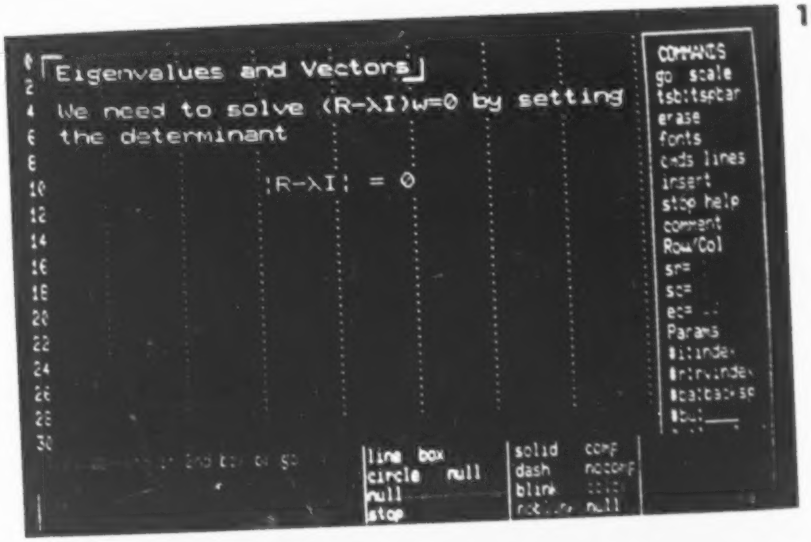


```
This is an example of the standard
text screen of 45 columns. Color can
be changed to green on the fly.

X^2_J superscripts and subscripts

Fonts: Σ λ ∂ Φ β Χ can be seen on the
the right......

[illegible] #i,#r,etc:cmds,fonts, or stop
```

9 Text mode with display of font set

10 Example of placing box around title

10


Eigenvalues and Vectors

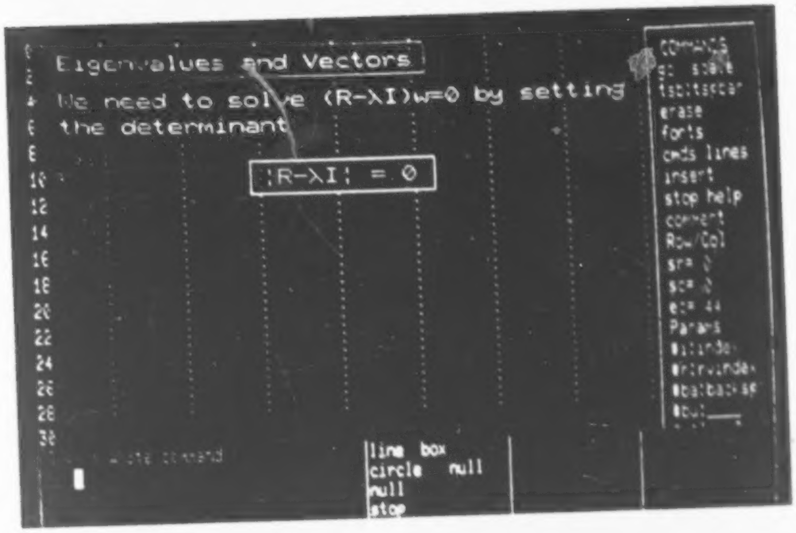
We need to solve (R-λI)w=0 by setting
the determinant

|R-λI| = 0

COMMANDS
go scale
tsb:tsptar
erase
fonts
cmds lines
insert
stop help
comment
Row/Col
sr=
sc=
ec=
Params
#i:index
#r:rvindex
#ba:backsp
#bu:
line box
circle null
null
stop
solid comp
dash nocomp
blink
null

11 Example with boxes around text completed

11


Eigenvalues and Vectors

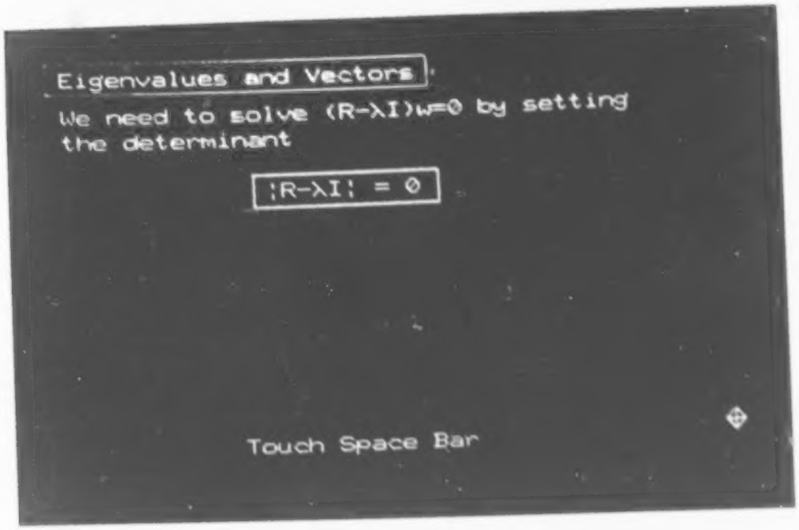
We need to solve (R-λI)w=0 by setting
the determinant

|R-λI| = 0

COMMANDS
go scale
tsb:tsptar
erase
fonts
cmds lines
insert
stop help
comment
Row/Col
sr= 0
sc= 0
ec= 44
Params
#i:index
#r:rvindex
#ba:backsp
#bu:
line box
circle null
null
stop

12 Effect actions in 10 and 11 as seen by student

12

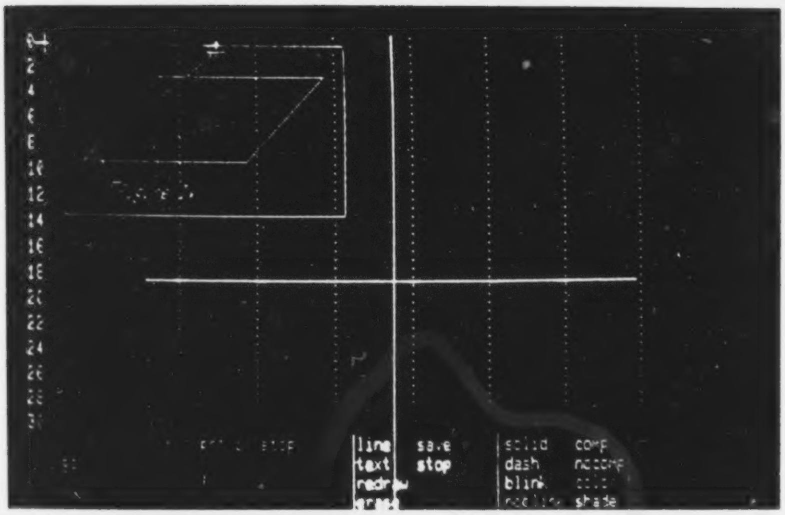

Eigenvalues and Vectors

We need to solve (R-λI)w=0 by setting
the determinant

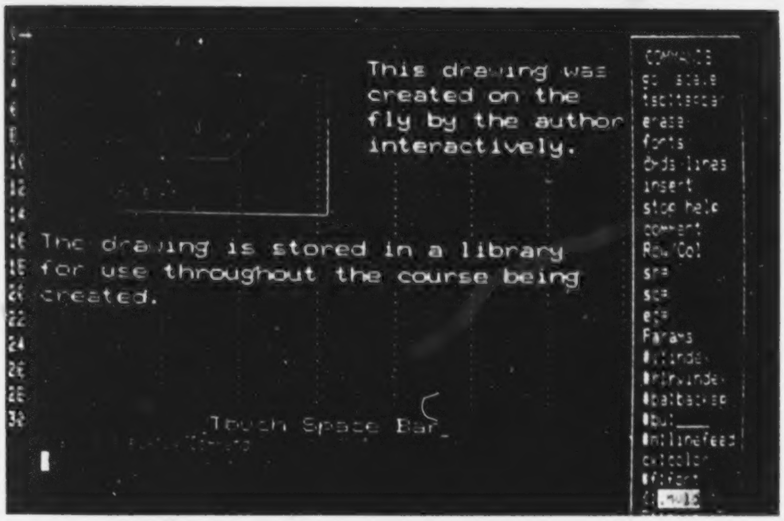
|R-λI| = 0

Touch Space Bar

13 Creation of drawing using cross hairs


13
line
text
redraw
save
stop
solid
dash
blink
comp
nocomp
shade

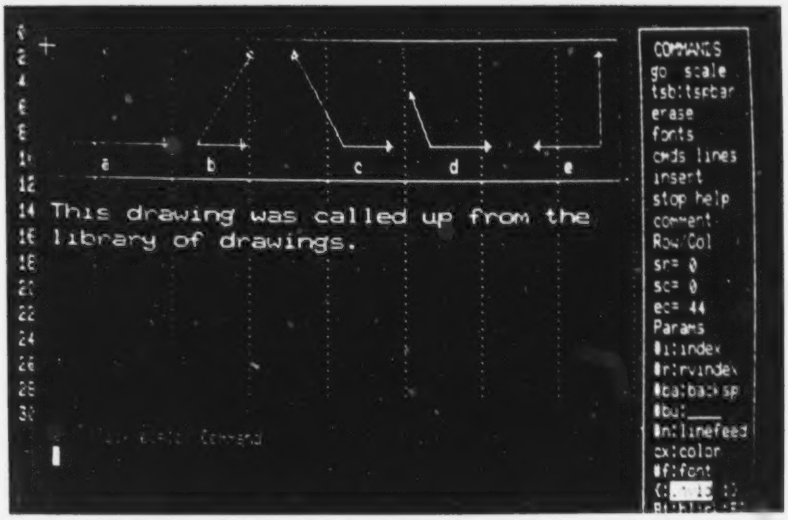
14 Drawing with surrounding text


This drawing was created on the fly by the author interactively.

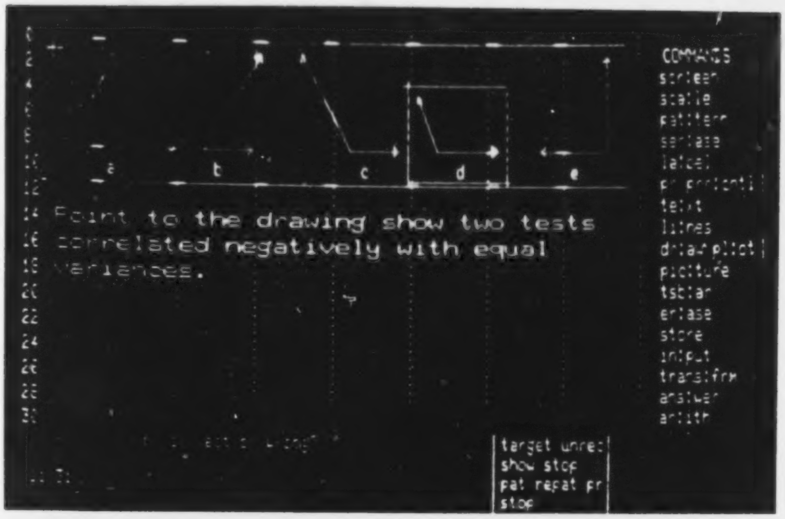
The drawing is stored in a library for use throughout the course being created.

Touch Space Bar
14
COMMANDS
go scale
erase
fonts
cmds lines
insert
stop help
comment
Row/Col
sr=
sc=
ec=
Params
#bu:____
#n:linefeed

15 Example of drawing called from library


This drawing was called up from the library of drawings.
15
a
b
c
d
e
COMMANDS
go scale
tsb:tspbar
erase
fonts
cmds lines
insert
stop help
comment
Row/Col
sr= 0
sc= 0
ec= 44
Params
#i:index
#r:rvindex
#ba:backsp
#bu:____
#n:linefeed
cx:color
#f:font

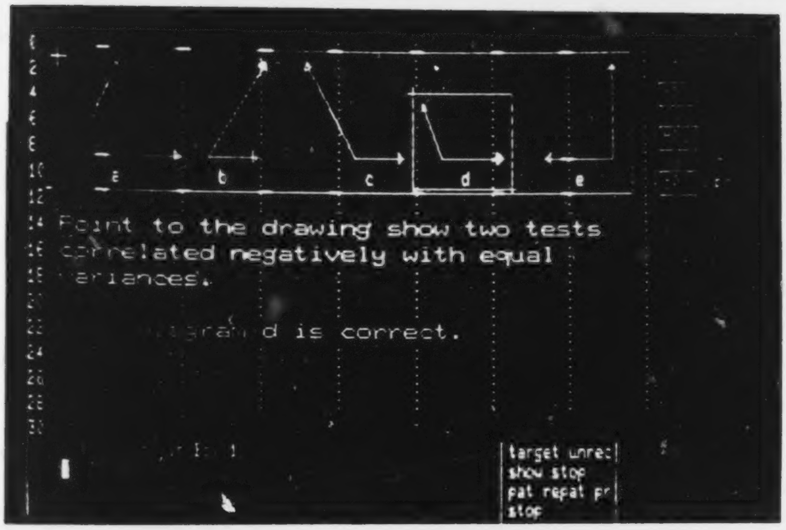
16 Defining a target area (d) for point response


a
b
c
d
e
COMMANDS
screen
scale
pattern
seriate
label
pr prrionti
text
lines
draw plot
picture
tsbar
erase
store
input
transfrm
answer
arith
target unrec
show stop
pat repat pr
stop

Point to the drawing show two tests correlated negatively with equal variances.

16

17 Example using defined target area as seen by author

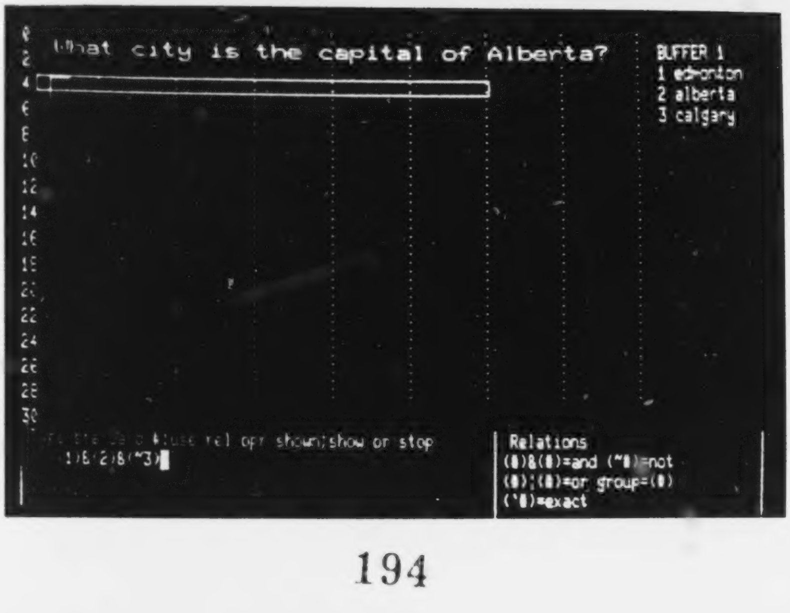

a
b
c
d
e
target unrec
show stop
pat repat pr
stop

Point to the drawing show two tests correlated negatively with equal variances.

Diagram d is correct.

17

18 Defining a Boolean standard for answer judging



What city is the capital of Alberta?

BUFFER 1
1 edmonton
2 alberta
3 calgary

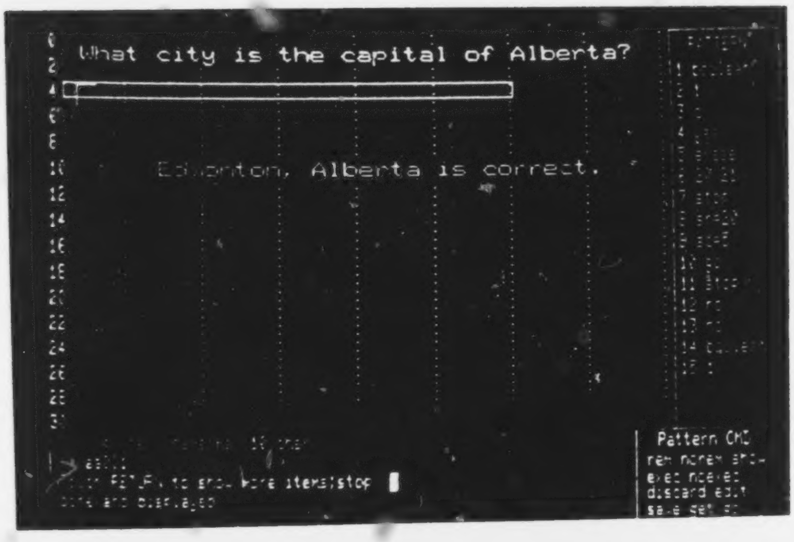
```
[illegible] use rel opr shown;show or stop
  (1)&(2)&(~3)
```

Relations
(#)&(#)=and (~#)=not
(#);(#)=or group=(#)
('#)=exact

18

19 Example of a pattern recalled for Boolean analysis

19



What city is the capital of Alberta?

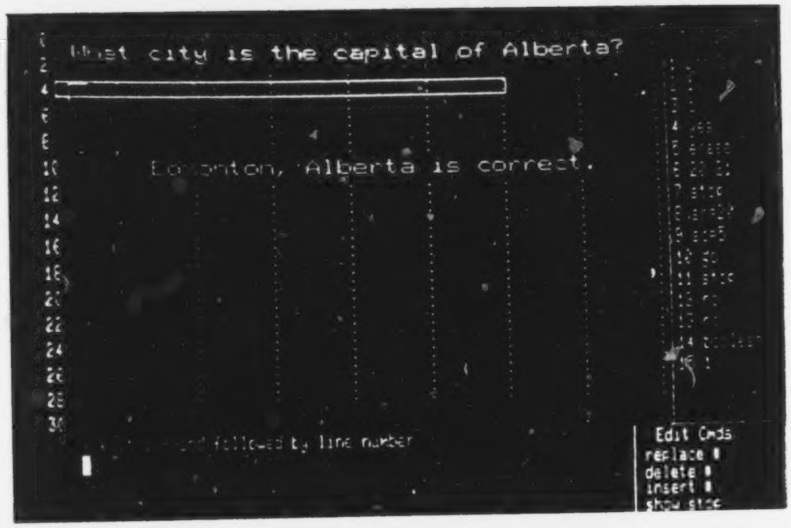
Edmonton, Alberta is correct.

RETURN to show more items;stop

Pattern CMD
rem norem
exec noexec
discard edit

20 Example of edit menu for use with pattern

20



What city is the capital of Alberta?

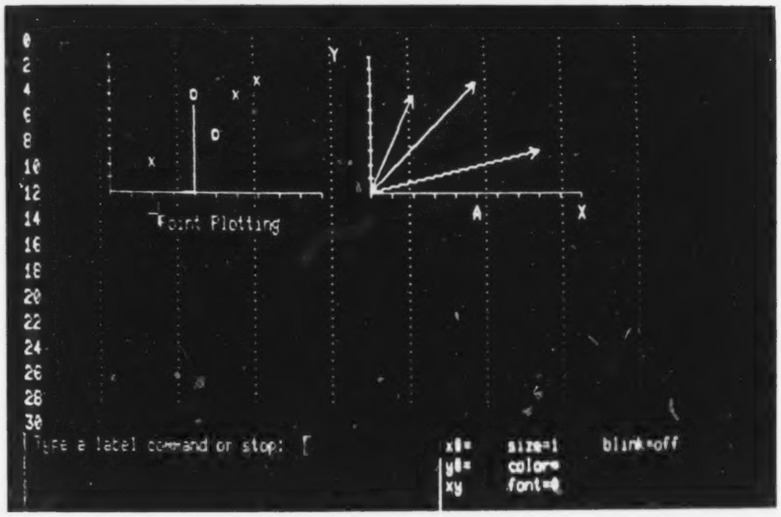
Edmonton, Alberta is correct.

followed by line number

Edit Cmds
replace
delete
insert
show stop

21 Examples of point and vector plotting at label stage

21

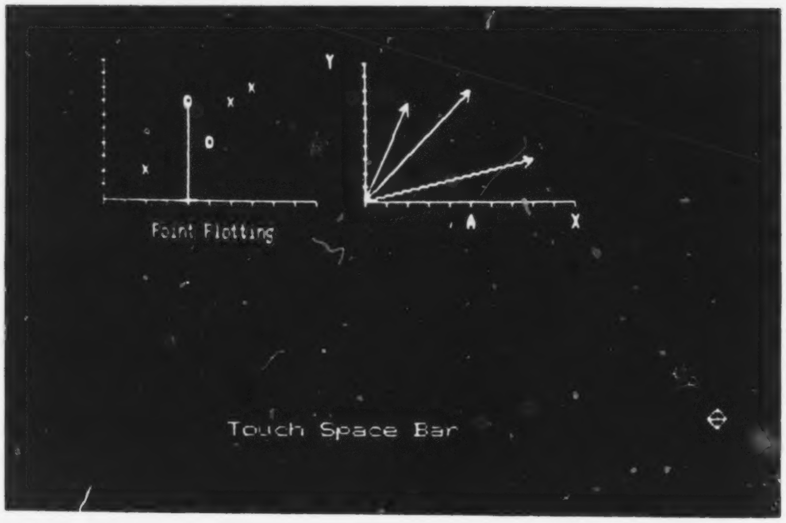


Y
O
X
X
O
X
Point Plotting
A
X

```
Type a label command or stop:
x0=    size=1    blink=off
y0=    color=
xy     font=0
```

22 Point and vector plots as seen by student

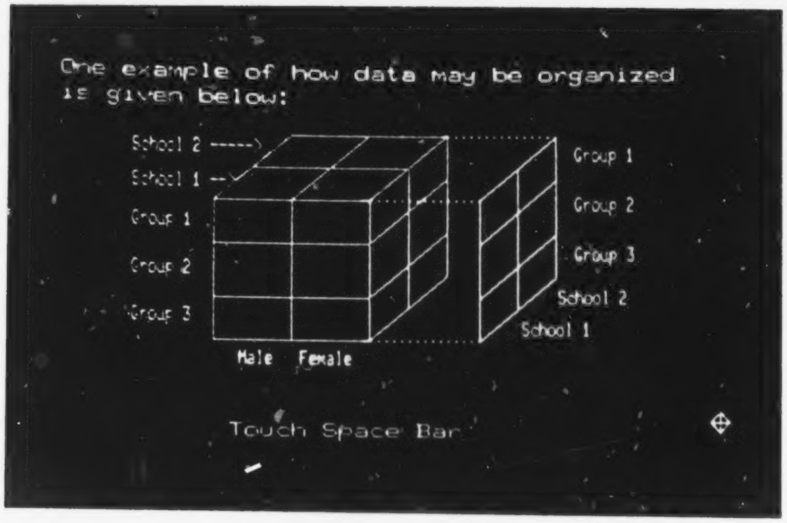
22


Y
Point Plotting
A
X

Touch Space Bar

23 Example of diagram as seen by student

23


One example of how data may be organized is given below:

School 2 ----->
School 1 -->
Group 1
Group 2
Group 3
Male
Female
Group 1
Group 2
Group 3
School 2
School 1

Touch Space Bar

research and information report

RIR-87-5

APPENDIX B

Author Manual for the DERS Interactive Experimental Authoring System (XAS)

SSHRC PROJECT NO. 410-84-1111-R2

S. Hunka

division of educational research
faculty of education
the university of alberta
edmonton, canada

RIR-87-5

APPENDIX B

Author Manual for the DERS Interactive Experimental Authoring System (XAS)

SSHRC PROJECT NO. 410-84-1111-R2

S. Hunka

Division of Educational Research Services
Faculty of Education
University of Alberta, Edmonton
February 1988

PREFACE

This authoring system was one of two systems worked on under a research grant provided by the Social Sciences and Humanities Research Council of Canada (Project 410-84-1111-R2). Their support has been very much appreciated and is gratefully acknowledged.

During the last couple of years a number of authoring systems have been developed. Many of these systems are designed to provide authoring capabilities in a restricted domain of instructional strategies. Some systems appear to have been designed for frame by frame presentation of questions, allowing little flexibility for creative use of instructional approaches. New authoring systems are likely to appear as part of the natural evolution of software capabilities. This is as it should be.

The primary purpose for the development of the authoring system described in this manual was to explore the problems associated with the author-terminal interface when the author is given a wide degree of freedom in the design of instructional approaches. The authoring system, has therefore, the following characteristics:

1. All authoring is done interactively using a keyboard and the cross hair feature of the DEC Gigi™ terminal. This includes the formatting of the screen with text as well as graphics.

2. The author creates text and drawings as they would be seen by the student.

3. There are no predefined instructional strategies or manner by which the screen must be formatted by the author. However, the author may create an unlimited number of templates of useful instructional sequences for repeated use to increase efficiency.

4. The authoring system was designed for the development of tutorial CAI. It is not a testing or computer-managed learning development system.

5. The authoring system does produce an internal code which is as far as is known, free from errors during execution.

6. A course created with the authoring system can be executed on the DERS CAI system with sufficient stability to warrant the scheduling of regular classes.

7. The author controls the actions of the authoring system by simple commands displayed through menus. The menus are context specific and hierarchical, usually to a depth of two.

8. In most instances, menus contain access to a help feature.

9. The development of the system, of course, reflects the constraints of the DEC Gigi terminal.

Table of Contents

Page

Section 1 1
XAS CAI AUTHORING SYSTEM MANUAL 1
1.0 Background 1
2.0 Cautions 1
3.0 About the Authoring System 2

Section 2 3
Program Code and Vax Use 3
1.0 Course Structure 3
2.0 Author System Processes 3
    2.1 Authoring Actions 3
3.0 VAX File System 4
4.0 Editing the Course 4

Section 3 5
Using the Authoring System 5
1.0 Preliminary Options 5
2.0 The First Display of The System 6

Section 4 10
Main Command Menu 10
1.0 Overview of Main Menu Commands 10

Section 5 15
A Typical Sequence of Authoring Steps 15

Section 6 16
Screen Formats 16
1.0 screen 16
2.0 45 Column Screen 16
3.0 Split Screen 16
4.0 80 Column Screen 16
5.0 Using 45 column and 80 Column Format Together 17

Section 7 18
Scale 18

Section 8 20
Pattern 20
1.0 Introduction 20
2.0 Pattern Options 20

Section 9 23
Label 23
1.0 Introduction 23
2.0 Label Options 24

Section 10 25
pr prr;cntl 25

Section 11 26
Text Commands 26
1.0 General Sequence 27
NOTE OF CAUTION
2.0 TEXT Commands and Parameters 28
3.0 ROW/COL 29
4.0 Params 30

Section 12 32
Lines 32
1.0 Introduction 32
2.0 Options 32

Section 13 34
Draw 34
1.0 Drawing Commands 35
2.0 Line Drawing Attributes 36
3.0 Marking Points on the Screen 37
4.0 Shading to the Y-axis 37
5.0 Labelling a Drawing 38

Section 14 40
Plot 40
1.0 Bar, Point, Continuous, and Vector Plots 40
2.0 Plot Prompt Sequence 41
    2.1 Preliminary Information 41
    2.2 Drawing and Marking the Axes 42
    2.3 Scaling Information 42
3.0 Entry of Data 43
4.0 Labelling the Axes 43

Section 15 44
Picture 44
1.0 Introduction 44
2.0 Author Query 45

Section 16 46
Erase 46

Section 17 47
Store 47
1.0 Background 47
2.0 The Store Options 47
3.0 Loading Labels 48

Section 18 50
Input 50
Introduction 50

Section 19 52
Transfrm 52
1.0 Introduction 52

Section 20 54
Answer 54
1.0 Introduction 54
2.0 General Procedure: Answer from Keyboard 55
3.0 Automatic Score Keeping Option 56
4.0 LIMIT Option 56
5.0 BOOLEAN Option 57
6.0 General Procedure: Pointing Response 58
7.0 UNREC Option 59
8.0 SHOW Option 59
9.0 Pat, Repat Options 60
10.0 pr 60

Section 21 61
Arith 61
1.0 Introduction 61

Section 22 64
Comment 64

Section 23 65
Branch 65

Section 24 67
Run 67
1.0 Introduction 67
2.0 Use of the RUN Option 67
3.0 Appending to an existing Segment 67
4.0 Merging new Content into an Existing Segment 68

Section 25 69
edit 69

Section 26 72
Exit 72

Section 27 73
Program Code Generated 73

Section 28 75
Execution of the Course Authored 75
1.0 Introduction 75

Section 29 76
Files Used by The Authoring System 76
1.0 Introduction 76
2.0 File Names 76
3.0 Files Used by The Authoring System 76
4.0 Authoring Program Files 77

Section 30 79
Using the External EDT Editor 79

Section 31 81

Startup of the Authoring System 81
1.0 Initialization of Files 81
2.0 Logical Assignment File 81
3.0 Down Loading Character Fonts 81
4.0 VAX Computer File Names 82
5.0 Operations Related to Files 82

Section 1
XAS CAI AUTHORING SYSTEM MANUAL

1.0 Background

The system described in this document provides the author with a facility to create a tutorial CAI course without directly needing to learn how to program. This is accomplished by allowing the author the capability of constructing program code for a course by interacting directly with a computer terminal in a "you get what you see" mode. In effect, the system writes the course program code by asking the author what needs to be done. There are no predetermined instructional strategies within which the author must operate.

The authoring system was written to support the development of large courses, such as would be taught during a semester.

This authoring system runs on the Digital Equipment VAX 11/780 computer located in the Division of Educational Research Services (DERS), with course delivery to 20 DEC Gigi terminals with color display monitors. Terminals for purposes of authoring can be located remotely. Since the software has been developed in DERS, specialized features can be created to accommodate specific authoring requirements.

This application package was developed to test out various concepts for use in the authoring system being developed under a research grant from the Social Sciences and Humanities Research Council. It has been used to construct approximately 15 hours of a matrix algebra course. In addition, approximately 30 undergraduate students used an earlier version of this system as part of a course on courseware authoring.

2.0 Cautions

The following inadequacies are noted:

- since output is an author language code, strategies of instruction, and to some extent the authoring procedures, are limited to what can be easily represented in program code.
- there may be "bugs" in the programs which are used.
- A basic tool of an authoring system is an adequate editor. A small editor is available which allows editing the course code generated by the authoring system, with results immediately observable. However, this editor is not as powerful as the editors resident in the VAX system.
- Since the number of paths through various authoring sequences is extensive, it is likely that some guiding messages to the author may be inadequate.

When important inadequacies are noted, these should be reported to Steve Hunka. For purposes of studying the procedures authors use, one file of information ("an audit trail file") is generated which records all the actions the author has taken. This file should not be destroyed, as it provides the exact path which an author has followed and becomes invaluable for investigating problems which have developed.

3.0 About the Authoring System

The authoring system has been programmed in a language called Elf (Educational Language Facility) designed by Alan Davis of DERS. The authoring system has approximately 24 separate programs (called grammars) each one responsible for a major authoring process. New authoring processes can be added to provide the author with unique capabilities which may be required for the subject matter for which the courseware is being created.

Section 2
Preliminaries
Program Code and Vax Use

1.0 Authoring System Model

This authoring system allows the construction of a course by creating on the screen what the author wishes the student to see. There are no predefined instructional strategies which must be followed. The use of an authoring system does not obviate the need for careful planning on the part of the author. Although excursions in presentations can be made, the logic of answer analysis and associated branching requirements should be carefully planned in advance of an authoring session.

2.0 Course Structure

The general structure of a course created by the authoring system is as follows:

1) A course is usually made up of chapters called segments. Segments are made unique by assignment of a 1-3 digit number e.g., stat4023 would identify the course stat4 and segment 023.

2) Within a course segment the author must define a number of entry points (labels). These are points in the course to which a branch can be made during course execution. It is the author's option where these are placed. However, when a new course is started the first three labels are automatically set to be the course name, the label "start" and the segment number followed by 000. These are set because the run time system may use these as defaults under certain conditions.

3) Course code is created and stored in a linear fashion by the authoring system. Drawings, however, are stored as separate named entities (files).

4) The last statement in a segment is "en" which signals the end of the segment to the interpreter. This code is automatically generated when leaving the authoring system.

2.1 Authoring Actions

A course is created by the author executing a number of authoring actions. Some actions result in screen effects, others do not.

Examples of authoring actions which result in observable screen effects during authoring and during course execution are as follows:

1) erase screen or portion thereof

2) place text on the screen using one or more fonts

3) create, or display a colored drawing

4) obtain an answer from the student, defining where the input will appear on the screen.

The elementary actions which do not directly create a screen effect are:

- establish labels or branch to points
- make arithmetic calculations
- check for a response over the time limit
- define a branch
- edit an input response
- check an answer

The author can easily observe the first set of actions, but must visualize or call upon support information for observing the second set of actions. However,the system does guide the author to provide the type of information required in order to create the actions which do not have an observable screen effect.

Depending upon the privilege level set for the author, new course code may be automatically appended to code from a previous session, or be created as an independent unit for later concatenation with existing code.

3.0 VAX File System

The authoring system makes use of files for the storage of course code, and that information which is required for continuation of authoring sessions. Depending upon the privilege level of the author, some understanding of the file system may be important. (See Section 29 for more information).

4.0 Editing the Course

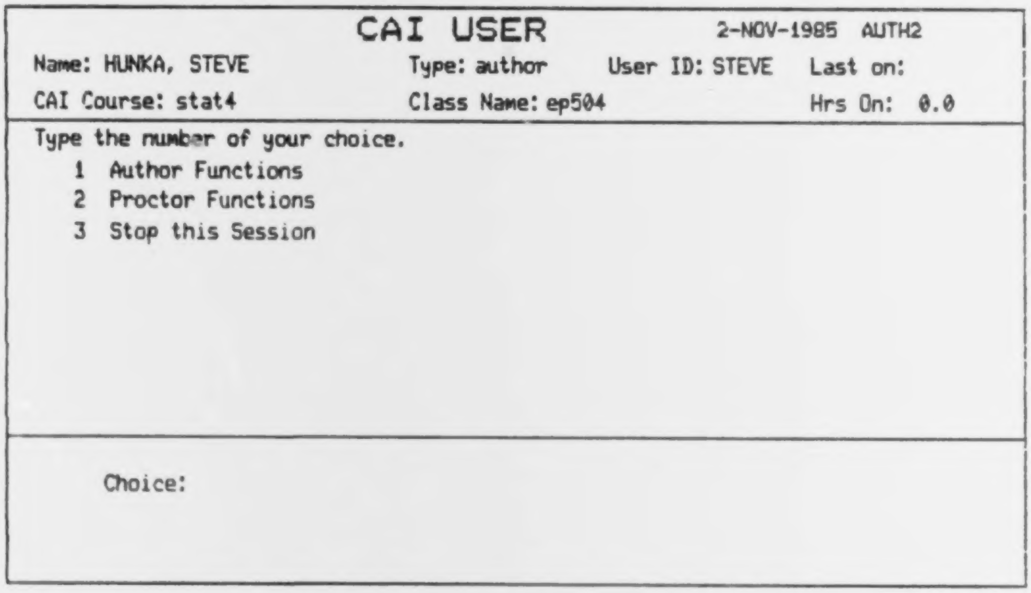
The file of course code created by the authoring system may be edited by an editor within the authoring system or by using an independent editor. The procedure for entering an independent editor will depend upon the environment established for the author at the time of registration. For example, an author may be allowed direct entry to an independent editor without having to carry out special operating system commands. (The use of the VAX EDT editor is described in Section 30).

Section 3
Using the Authoring System

1.0 Preliminary Options

The exact manner an author enters the authoring system may vary depending upon the extent to which the author is allowed access to the VAX operating system. For example, student authors are usually provided with a more direct entry to the authoring system, and without access to the operating system commands.

Authors having access to the VAX operating system, may enter the authoring system by typing the command "elf" followed by the command "ex cw7$cai". These steps allow entry into the authoring sequence by providing the author with a series of menus. The first menu shown below, requests whether author or proctor functions are required (select Author Functions).



```
                              CAI USER            2-NOV-1985  AUTH2
Name: HUNKA, STEVE            Type: author    User ID: STEVE   Last on:
CAI Course: stat4             Class Name: ep504                Hrs On:  0.0

Type the number of your choice.
   1  Author Functions
   2  Proctor Functions
   3  Stop this Session

      Choice:
```

The second menu shown below lists a series of processes available to the author. Select number 2 (SSHRC Authoring System).

Section 1
XAS CAI AUTHORING SYSTEM MANUAL

1.0 Background

The system described in this document provides the author with a facility to create a tutorial CAI course without directly needing to learn how to program. This is accomplished by allowing the author the capability of constructing program code for a course by interacting directly with a computer terminal in a "you get what you see" mode. In effect, the system writes the course program code by asking the author what needs to be done. There are no predetermined instructional strategies within which the author must operate.

The authoring system was written to support the development of large courses, such as would be taught during a semester.

This authoring system runs on the Digital Equipment VAX 11/780 computer located in the Division of Educational Research Services (DERS), with course delivery to 20 DEC Gigi terminals with color display monitors. Terminals for purposes of authoring can be located remotely. Since the software has been developed in DERS, specialized features can be created to accommodate specific authoring requirements.

This application package was developed to test out various concepts for use in the authoring system being developed under a research grant from the Social Sciences and Humanities Research Council. It has been used to construct approximately 15 hours of a matrix algebra course. In addition, approximately 30 undergraduate students used an earlier version of this system as part of a course on courseware authoring.

2.0 Cautions

The following inadequacies are noted:

- since output is an author language code, strategies of instruction, and to some extent the authoring procedures, are limited to what can be easily represented in program code.
- there may be "bugs" in the programs which are used.
- A basic tool of an authoring system is an adequate editor. A small editor is available which allows editing the course code generated by the authoring system, with results immediately observable. However, this editor is not as powerful as the editors resident in the VAX system.
- Since the number of paths through various authoring sequences is extensive, it is likely that some guiding messages to the author may be inadequate.

When important inadequacies are noted, these should be reported to Steve Hunka. For purposes of studying the procedures authors use, one file of information ("an audit trail file") is generated which records all the actions the author has taken. This file should not be destroyed, as it provides the exact path which an author has followed and becomes invaluable for investigating problems which have developed.

3.0 About the Authoring System

The authoring system has been programmed in a language called Elf (Educational Language Facility) designed by Alan Davis of DERS. The authoring system has approximately 24 separate programs (called grammars) each one responsible for a major authoring process. New authoring processes can be added to provide the author with unique capabilities which may be required for the subject matter for which the courseware is being created.

Section 2
Preliminaries
Program Code and Vax Use

1.0 Authoring System Model

This authoring system allows the construction of a course by creating on the screen what the author wishes the student to see. There are no predefined instructional strategies which must be followed. The use of an authoring system does not obviate the need for careful planning on the part of the author. Although excursions in presentations can be made, the logic of answer analysis and associated branching requirements should be carefully planned in advance of an authoring session.

2.0 Course Structure

The general structure of a course created by the authoring system is as follows:

1) A course is usually made up of chapters called segments. Segments are made unique by assignment of a 1-3 digit number e.g., stat4023 would identify the course stat4 and segment 023.

2) Within a course segment the author must define a number of entry points (labels). These are points in the course to which a branch can be made during course execution. It is the author's option where these are placed. However, when a new course is started the first three labels are automatically set to be the course name, the label "start" and the segment number followed by 000. These are set because the run time system may use these as defaults under certain conditions.

3) Course code is created and stored in a linear fashion by the authoring system. Drawings, however, are stored as separate named entities (files).

4) The last statement in a segment is "en" which signals the end of the segment to the interpreter. This code is automatically generated when leaving the authoring system.

2.1 Authoring Actions

A course is created by the author executing a number of authoring actions. Some actions result in screen effects, others do not.

Examples of authoring actions which result in observable screen effects during authoring and during course execution are as follows:

1) erase screen or portion thereof

2) place text on the screen using one or more fonts

3) create, or display a colored drawing

4) obtain an answer from the student, defining where the input will appear on the screen.

The elementary actions which do not directly create a screen effect are:

- establish labels or branch to points
- make arithmetic calculations
- check for a response over the time limit
- define a branch
- edit an input response
- check an answer

The author can easily observe the first set of actions, but must visualize or call upon support information for observing the second set of actions. However,the system does guide the author to provide the type of information required in order to create the actions which do not have an observable screen effect.

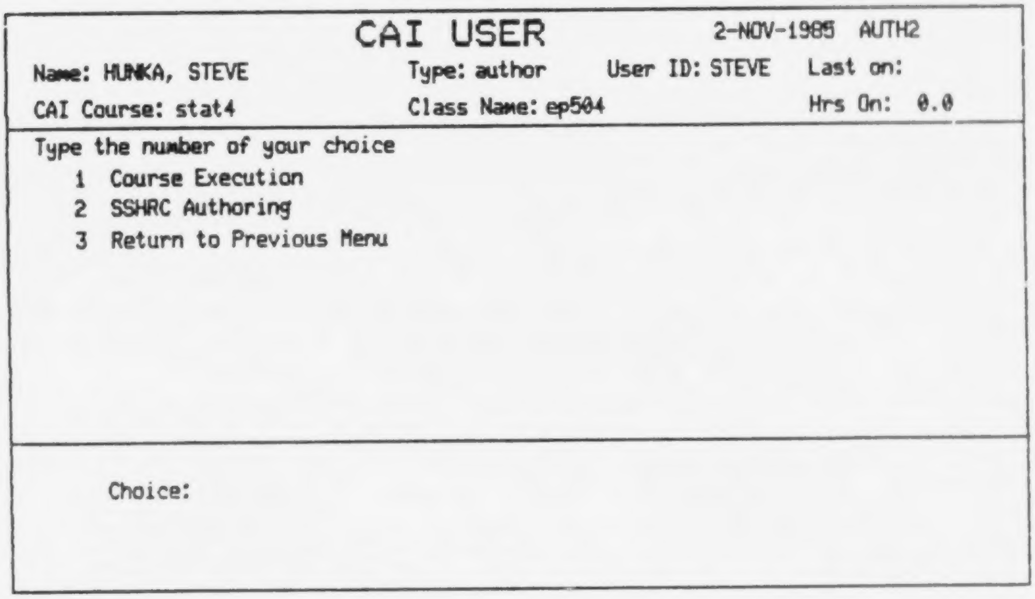
Depending upon the privilege level set for the author, new course code may be automatically appended to code from a previous session, or be created as an independent unit for later concatenation with existing code.

3.0 VAX File System

The authoring system makes use of files for the storage of course code, and that information which is required for continuation of authoring sessions. Depending upon the privilege level of the author, some understanding of the file system may be important. (See Section 29 for more information).

4.0 Editing the Course

The file of course code created by the authoring system may be edited by an editor within the authoring system or by using an independent editor. The procedure for entering an independent editor will depend upon the environment established for the author at the time of registration. For example, an author may be allowed direct entry to an independent editor without having to carry out special operating system commands. (The use of the VAX EDT editor is described in Section 30).



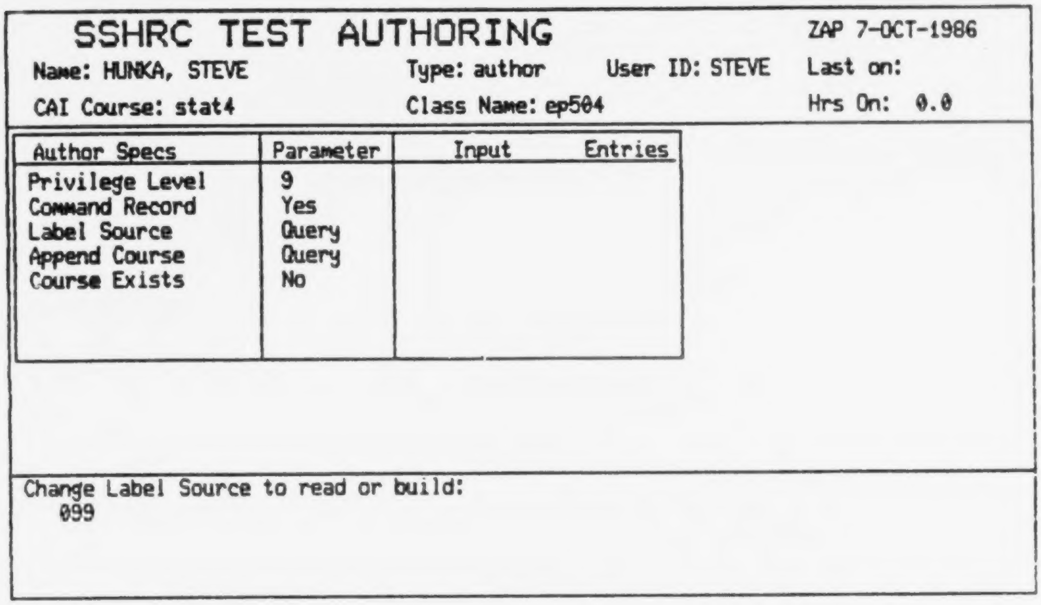
```
                      CAI USER               2-NOV-1985  AUTH2
Name: HUNKA, STEVE            Type: author     User ID: STEVE   Last on:
CAI Course: stat4             Class Name: ep504                 Hrs On:  0.0

Type the number of your choice
    1  Course Execution
    2  SSHRC Authoring
    3  Return to Previous Menu



        Choice:
```

Authors with privilege level 9, are then required to enter the 3 digit number of the segment or chapter they wish to construct. Student authors with privilege level 0 are automatically assigned authorship of segment 000.

From this point on, the author is within control of the authoring system. The next few sections illustrate some of the menus which the author will encounter.

2.0 The First Display of The System

A small table is presented to the author summarizing the author's parameters (Author Specs) which will govern the current authoring session corrected for any logical inconsistencies, e.g., the current session's course code cannot be appended to a previous session's course code if the latter does not exist. This table is constructed from the parameters set in a special file of information defined at the time the author is registered. An example of the display follows.



```
SSHRC TEST AUTHORING                                ZAP 7-OCT-1986
Name: HUNKA, STEVE          Type: author   User ID: STEVE   Last on:
CAI Course: stat4           Class Name: ep504               Hrs On:  0.0

 Author Specs      | Parameter |  Input      Entries
 Privilege Level   | 9         |
 Command Record    | Yes       |
 Label Source      | Query     |
 Append Course     | Query     |
 Course Exists     | No        |

Change Label Source to read or build:
   099
```

Privilege Level: this is a number in the sequence from 0 to 9 which governs the extensiveness of the functions the author may carry out.

Command Record: this will contain either yes or no. If the parameter is yes, keyboard actions of the author will be saved; if no, the keyboard actions will not be saved. The saving of keyboard actions provides the basis for isolating errors in the authoring system.

Label Source: The parameters may be read or build. If the parameter is set to read, the labels created for the course being authored will be read from a file created at the last authoring session; if the parameter is build, the labels will be created by scanning the course code created in the last authoring session. If the author wishes to be assured that the labels truly exist then the parameter build should be used.

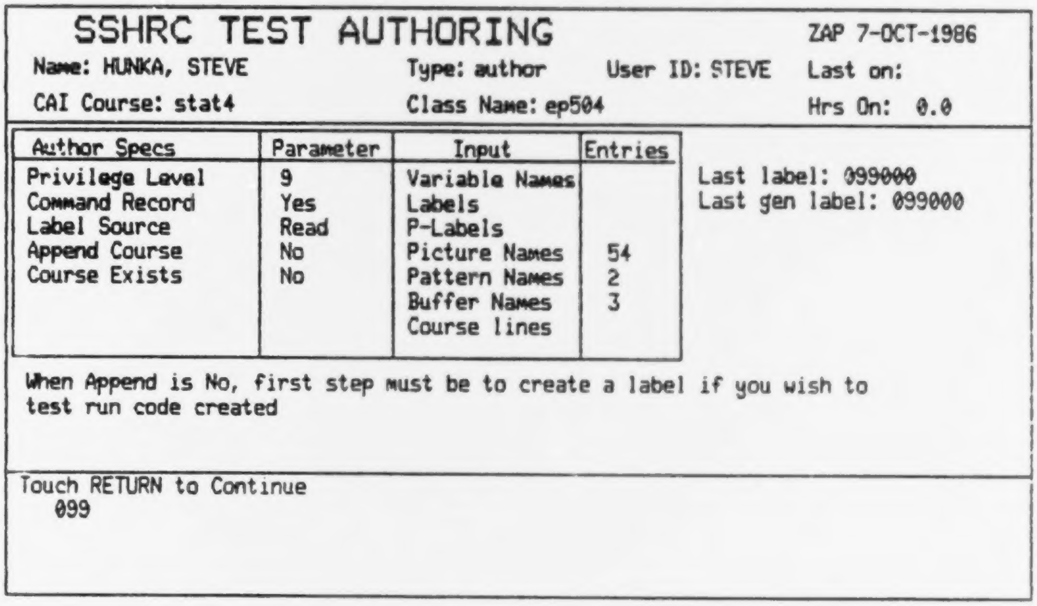
Append Course: The parameters here may be yes or no. If the parameter is set to yes, the work of the current session will be appended to the end of the course code which already exists for the segment being authored; if set to no, the current session's code will not be appended to existing code of the same segment. Care must be excerised when this is done as two files for the same segment will exist after authoring.

Course Exists: The parameters here will be yes or no. If the system finds that a course segment already exists with the same number as the one to be authored, this parameter is set to yes; otherwise it is set to no.

In the case of an author with privilege level 9, any parameter having the term "query" associated with it, will cause the system to request the author to replace the term. This allows the author to tailor the parameters for the needs of the current session. The new settings are also checked for logical inconsistencies, e.g., if no course exists Append Course is set to no and Label Source is set to read.

When a course is authored for the first time the first label and second label are automatically set to the name of the course, and "start", and the first all numeric label made up of the segment number followed by 000 is created. Therefore, the author will not be allowed to create these labels during an authoring session.

A second table containing information about the course to be authored is illustrated below:



```
    SSHRC TEST AUTHORING                                  ZAP 7-OCT-1986
  Name: HUNKA, STEVE              Type: author   User ID: STEVE   Last on:
  CAI Course: stat4               Class Name: ep504               Hrs On:  0.0

 Author Specs        Parameter   Input             Entries
 Privilege Level     9           Variable Names                Last label: 099000
 Command Record      Yes         Labels                        Last gen label: 099000
 Label Source        Read        P-Labels
 Append Course       No          Picture Names      54
 Course Exists       No          Pattern Names      2
                                 Buffer Names       3
                                 Course lines

 When Append is No, first step must be to create a label if you wish to
 test run code created

Touch RETURN to Continue
  099
```

Variable Names: The number of variables created in the course will be displayed

Labels: The number of labels currently being used in the course segment will be displayed

P-Labels: The number of labels in the segment to which a branch has been defined, but for which the branch to point has not yet been specified in the course, is displayed.

Picture Names: The number of drawings which have been created for the whole course (all segments) is displayed.

Pattern Names: The number of patterns which have been created by the author in the process of authoring any segment. (Patterns are templates created and named by the author).

Buffer Names: The number of buffer names used by the author in the course.

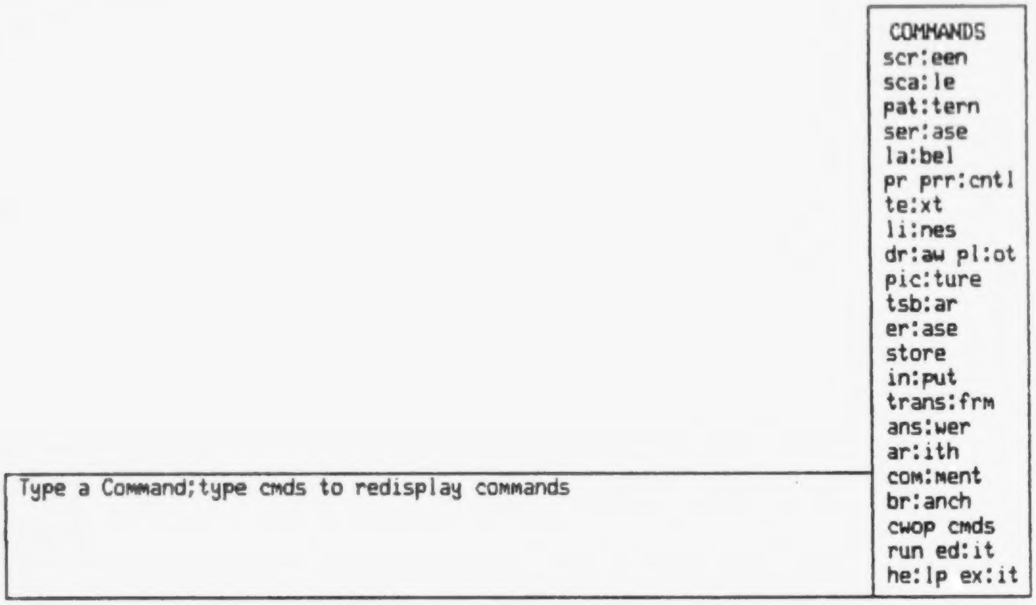
Course Lines: The number of lines of course code found in the segment which is to be authored.

The purpose of providing the two tables previously described is to tailor some of the authoring functions to the author's needs, and to provide supporting information, usually confirming or disconfirming, that the proper segment of the course is being authored. These may not be important to the author creating a trivial course sequence of a couple of hours or less, but become important when an author is working on 20 to 30 different segments of the same course.

Upon touching the RETURN key, the main menu containing authoring commands is then displayed to the author.

Section 4
Main Command Menu

The purpose of this section is to provide the author with a brief overview of the commands available from the main menu. The main menu is the menu from which all major authoring functions are called. Specific details about each command appear as separate sections.


COMMANDS
scr:een
sca:le
pat:tern
ser:ase
la:bel
pr prr:cntl
te:xt
li:nes
dr:aw pl:ot
pic:ture
tsb:ar
er:ase
store
in:put
trans:frm
ans:wer
ar:ith
com:ment
br:anch
cwop cmds
run ed:it
he:lp ex:it

```
Type a Command;type cmds to redisplay commands
```

Note that a box on the right-hand side of the screen contains a set of commands. Some commands assist the authoring process without generating course program code, some may generate code directly, while others will lead the author to other menus containing a new set of commands.

If a short-form of a command is available, it is indicated by the characters preceding the colon, eg., te:xt indicates that te or text may be used. In some cases the characters following the colon are explanatory. Parsing is always made for the characters before the colon.

It should be noted that the author may type quickly and ahead of the system accepting and acting upon the commands, because a "type ahead buffer" is active within the authoring system.

1.0 Overview of Main Menu Commands

scr:een

The screen command allows the author to select one of 3 screen column formats. The settings are a 45 column screen, a 40 column screen beginning on the left and ending about the middle of the screen, and an 80 column screen. The 45 column and 80 column screen formats occupy most of the screen, and therefore the 45 column format uses larger character sets than the 80 column screen.

The 45 column screen is typically very easy to read, but may be too restrictive to the author. For example when a complex arithmetic derivation is being placed on the screen the 45 column format is frequently too small necessitating an undesirable continuation of the deriviation on a second line. (See Section 6).

sca:le

This option provides a scale of digits 0-30, in steps of 2 units on the left side of the screen and vertical dotted lines marking columns 5, 10, 15, 20, 25, etc. The purpose of the scale is to help the author to align text and drawings placed on the screen. (See Section 7).

pat:tern

This option allows the author to save a sequence of commands for automatic re-execution. For example, in creating an answer analysis procedure, frequently the sequence would be to select the type of analysis, define the answer standard, define its categorization (right, wrong), erase a portion of the screen for a message, and to make a branch. This sequence can be saved and re-executed to re-create the moves, and during re-execution, allow the author to enter unique information, eg., reprise message specific to a particular answer given by the student. The pattern of commands can be named, saved, recalled, and edited. Thus, an author may define any instructional sequence (a template) for repeated use in different sections of the course. (See Section 8).

ser:ase

This option provides for the erasure of the vertical guidelines created under the scale option. This option is useful when a screen is made to contain both 45 column and 80 column formats.

la:bel

This option provides the facility to create a new label, to list existing labels, and to see attached author notes. See Section 9 for a further explanation of labels.

pr prr:cntl

The pr and prr command places pr and prr in the program code file. The pr and prr act as control points to which implicit branching may occur.

If the author wishes the pr to also become a restart entry point then a prr should be typed. In this case the prr defines the point at which the student will be restarted at the next signon. This is likely to be, for example the start of a new topic, a review, an introduction, or a new problem in a drill sequence. (See Section 10).

te:xt

Text provides entry into the system for placing text to the screen. A menu is provided on the right of the screen. The menu provides for cursor positioning, definition of character attributes, special operations, font changes, and screen erasures. (See Section 11).

li:nes

This facility was provided to draw lines, boxes, and circles on the screen. Line attributes are color, the 'dash' pattern, and complement. The author may create lines, boxes, and circles on a screen containing text, as might be required to draw the student's attention to a particular part of an equation. (See Section 12).

dr:aw

Draw provides entry to a drawing facility. The author will be queried as to the type of drawing drw or fp; the drw type is relocatable to any position on the screen, while the fp type will appear only on the right hand side of the screen under the "split" screen option. The author may also modify existing drawings by adding to them. All drawings are saved for repeated use. Drawing names are kept in a list with comments for author reference. (See Section 13).

pl:ot

This option provides the author with plotting or graphing capabilities for instructional purposes. The following are available: histogram or bar graph, plotting points, plotting continuous lines, and vectors. All plots are made in a 2 dimensional coordinate space. The author may enter data from the keyboard or have it read from a file. The plots are named and saved in the same way as drawings. (See Section 14).

pic:ture

Picture provides for the display on the screen of drawings which have already been created. These drawings may be of the drw or fp type. The drw type drawings are fully relocatable, while the fp type are fixed for display on the right side of the screen for use under the split screen format. (See Section 15).

tsb:ar

tsb provides the message Touch Space Bar on the screen in row 29 and 30. This "holds" the screen for the student to read until any key (a-z, 0-9, RETURN, ENTER, space bar) is touched.

er:ase

This command generates a screen erasure. Erasures may be defined by line numbers to erase over the full screen width, by default to erase the whole screen, or for a rectangular area of the screen. (See Section 16).

store

The store command is used to provide the author with the facility of storing keyword strings for purpose of answer analysis, and labels to which branches can be made during execution (indirect addressing). The keyword strings may also be loaded to any one of 6 'buffer' storage areas each containing 100 character positions for use during course execution. The display of buffer contents is available as an insert option under the text option of the main menu. (See Section 17).

in:put

The input command allows the author to specify the position on the screen where the cursor will appear for purposes of the student typing in an answer, or to define the process for accepting a pointing response. In the case of a keyboard input, the row and column position of the cursor is calculated from a rectangular area specified for input by the author using the cross hairs. (See Section 18).

trans:frm

Presently, this command allows the author to specify a string which is to be searched for in the student's reply, and to have it replaced by some other defined string. This provides for simple editing capability before the student's answer is processed by answer analysis algorithms. However, some of the answer analysis procedures already have such editing (eg., removal of leading and trailing spaces) as part of their definition. (See Section 19).

ans:wer

Since the input command allows asking for either a keyboard or a pointing response, the author is guided through procedures for analyzing both types of responses. In the case of a keyboard response, two methods are provided for checking the answer given by the student. (See Section 20).

ar:ith

arith provides access to the definition of variable names and calculation operations to be used in simple arithmetic operations. The calculations are made in integer form (a temporary restriction of the interpreter). The author is first required to define the name of the variable he wishes to use, eg., right, wrong, total. Expressions such as right=right+1, total = right + wrong, may then be used. (See Section 21).

com:ment

This command can serve a number of different functions because it stores an author's comment in the course code generated. The form of the line generated is: no #COM:author-comment. The author may use it to insert a messsge to indicate where a correction is required during the editing phase, or to make a comment about the course content. (See Section 22).

br:anch

This command is provided in the main menu and is also made available as part of the answer analysis procedure. The author is queried as to the type of branch desired (unconditional or conditional), the label to which the branch is to be made, and if a conditional branch, the variables and or constants involved and their relationship. (See Section 23).

cwop

In order to allow the author to enter commands which cannot yet be created by the authoring system, the CWOP command is provided. The opcodes acceptable are as follows: dl (underline), pa (pause) ca, cb, wa, aa, un, ea (all used in Coursewriter-type answer analysis), and fn (for using a whole series of special functions which are available for an assorted set of processes). Normally, an author does not need to use this option.

cmds

This command will cause the author's list of commands to be re-displayed on the right-hand side of the screen. This will first cause the command menu area to be erased, therefore a drawing in this area may have its right-most area destroyed during the authoring process. However it would appear correctly for the student.

run

The run command allows the author to execute the course code created by the authoring system. It also allows the author to insert a block of newly created code. (See Section 24).

ed:it

The editor for editing the code produced is invoked by typing ed or edit. The author is asked to define the segment and starting label of the code to edit. The segment may be a current segment just created, or a different segment. Each line of the code is displayed, and if it causes a screen effect, it is executed. The execution allows the author to see the code's current action as well as the action after editing. (See Section 25).

he:lp

The help option provides some guidance to the use of the commands in the menu.

ex:it

This option returns the author to the last menu, which in this case takes the author out of the authoring system. (See Section 26).

Section 5
A Typical Sequence of Authoring Steps

A list of the simplest steps for creating a tutorial sequence for presenting text and a question, getting a response, analyzing the response, and giving the student a reprise message is illustrated below:

1) **label:** create a label for the start of the sequence

2) **screen:** set the desired screen format; if not done the default is the 45 column screen

3) **erase:** erase the whole screen (this will automatically cause the scaling of the screen).

4) **draw or picture:** create or display a drawing

5) **text:** enter text-to-screen mode; present material to read or a question

6) **store:** store any keywords for answer analysis

7) **input:** define the type of input required

8) **transform:** select the type of transformation for editing the student input; if boolean answer analysis is to be used, this option must be selected

9) **answer:** carry out the answer analyses required (correct, wrong, unrecognized); an option to give a reply to the student is made available and if selected, then the author is placed in text-to-screen mode.

10) **erase:** erase the area of the screen where your reply will be given; type the reply to the student

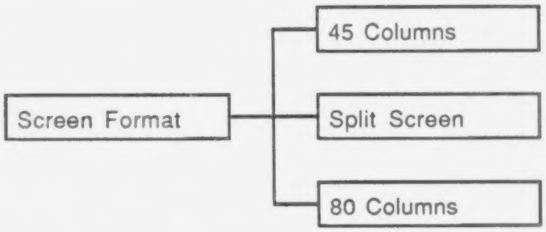
11) if desired the author may then enter the arithmetic mode to do simple score keeping (this can be set to be automatic)

12) if desired the author may specify branches, or take the default to branch ahead if the answer is identified as correct, or to branch back for a repeat of the input (normal case) if the answer was wrong or unrecognized.

13) **tsb:** causes screen to be held for the student.

Section 6
Screen Formats

1.0 screen



The command screen provides the author with selection of 3 different screen formats: a 45 column screen with large character fonts, a 40 column split screen with the left side having 40 columns and the right side blank for the display of diagrams, and an 80 column screen. The default specification is the 45 column screen format. For all 3 screen formats, 32 rows are provided which can hold 16 lines of text. When this command is executed there is no observable effect on the screen; the effect is shown when the command "scale" is given (See Section 7).

When varying formats of screens are used, the author must consider the implications of a student moving about in the course and that the appropriate screen is defined before text is presented. If the text for an 80 column format is displayed when the screen is set for 45 columns, text will be lost. Defined re-entry points (labels) to which a student might branch, must be followed by the appropriate screen format specification.

2.0 45 Column Screen

The 45 column screen uses most of the display area and uses a large character size.

3.0 Split Screen

"split" splits the screen into two parts in anticipation of the right-hand side being used to display a fixed position drawing. This command is normally followed by the command scale to provide the correct scaling guides for the screen. The split screen command should normally be used after a complete screen erasure has been made of the full-sized screen, otherwise the text from the full-sized screen may exist in the area reserved for fixed position drawings. Erasure when the split screen is current, will only erase the left-hand side of the screen, which contains 40 column positions for text. The left-hand side of the split screen uses a smaller character size than the 45 column format.

4.0 80 Column Screen

"80" provides for an 80 column screen using a small font size. This is useful when considerable text or long equations are required, e.g, equations in a derivation.

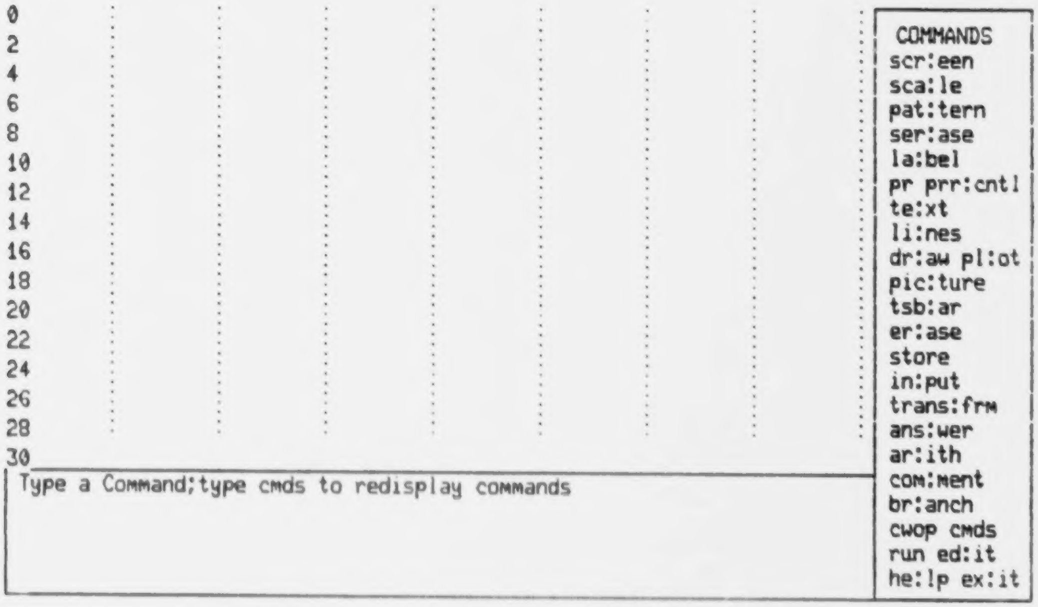
5.0 Using 45 column and 80 Column Format Together

The author may, for example, first define a 45 column screen, and then redefine the screen to an 80 column format so that in effect two sizes of text appear on the screen at the same time. The ser:ase command of the main menu would be used to erase the vertical guidelines of the 45 column format before invoking and scaling the 80 column format.

Section 7
Scale

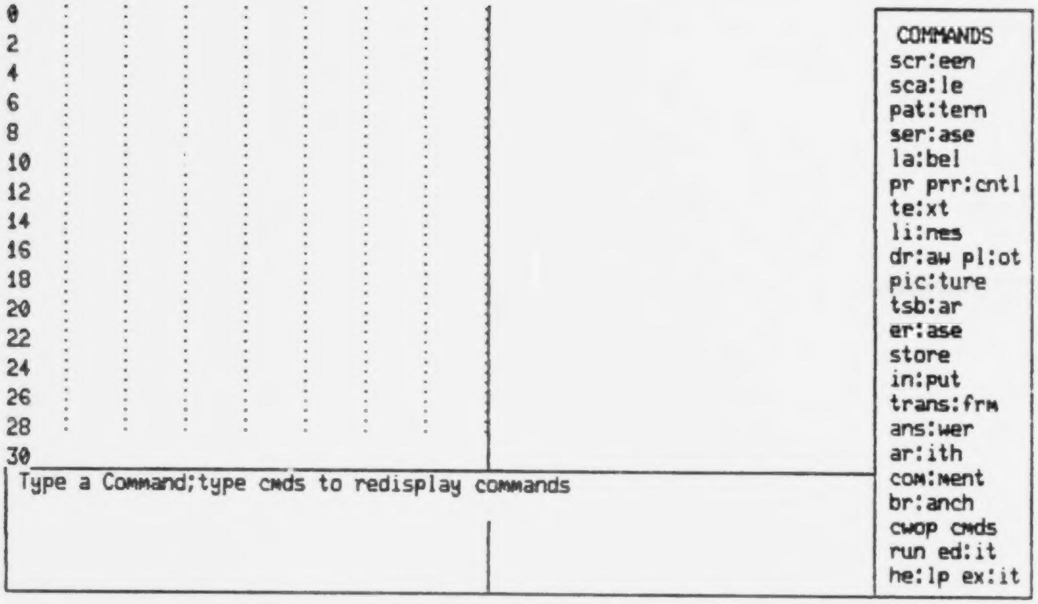
The **sca:le** option places line numbers down the left side of the screen and vertical guidelines across the screen in steps of 5 columns. This feature is provided to help guide the author in the placement of text and drawing on the screen.

Text may be placed beyond the scaled lines, however, this may interfere with the command box on the right. (Use command cmds to re-write the command box). Text will, however, be correctly displayed to the student, ie., without any problem of text not being correctly displayed or erased. It is generally a good idea while creating a screen display not to type into the command menu area on the right side of the screen. This will allow a few extra columns in case a word needs to be added to the line during the editing phase of course creation. The three types of screens available with their scaling are shown below.


0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
COMMANDS
scr:een
sca:le
pat:tern
ser:ase
la:bel
pr prr:cntl
te:xt
li:nes
dr:aw pl:ot
pic:ture
tsb:ar
er:ase
store
in:put
trans:frm
ans:wer
ar:ith
com:ment
br:anch
cwop cmds
run ed:it
he:lp ex:it

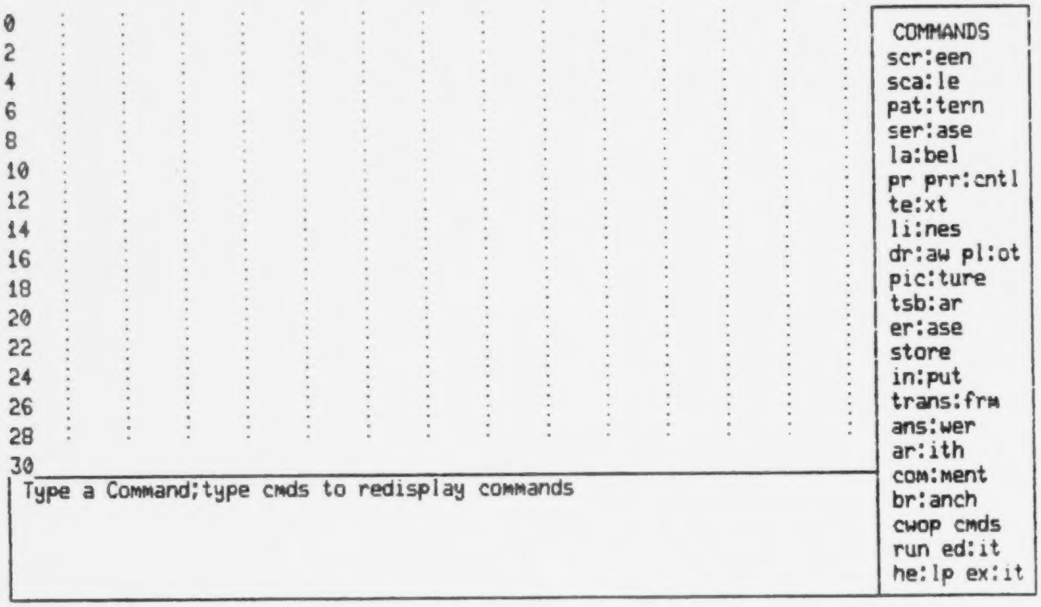
```
Type a Command;type cmds to redisplay commands
```

45 COLUMN SCREEN FORMAT


0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
COMMANDS
scr:een
sca:le
pat:tern
ser:ase
la:bel
pr prr:cntl
te:xt
li:nes
dr:aw pl:ot
pic:ture
tsb:ar
er:ase
store
in:put
trans:frm
ans:wer
ar:ith
com:ment
br:anch
cwop cmds
run ed:it
he:lp ex:it

```
Type a Command;type cmds to redisplay commands
```

SPLIT SCREEN FORMAT

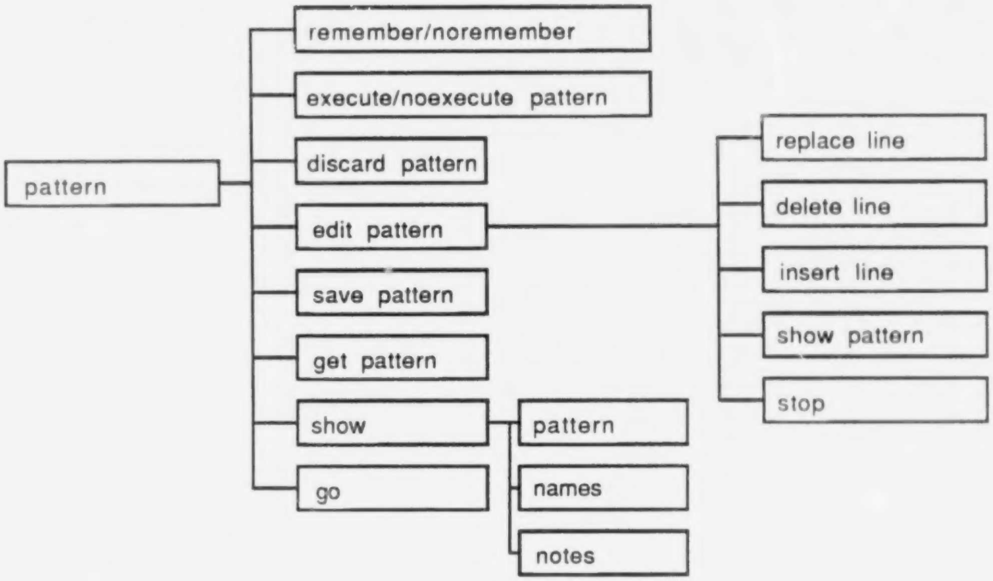

0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
COMMANDS
scr:een
sca:le
pat:tern
ser:ase
la:bel
pr prr:cntl
te:xt
li:nes
dr:aw pl:ot
pic:ture
tsb:ar
er:ase
store
in:put
trans:frm
ans:wer
ar:ith
com:ment
br:anch
cwop cmds
run ed:it
he:lp ex:it

```
Type a Command;type cmds to redisplay commands
```

80 COLUMN SCREEN FORMAT

Section 8
PATTERN

1.0 Introduction

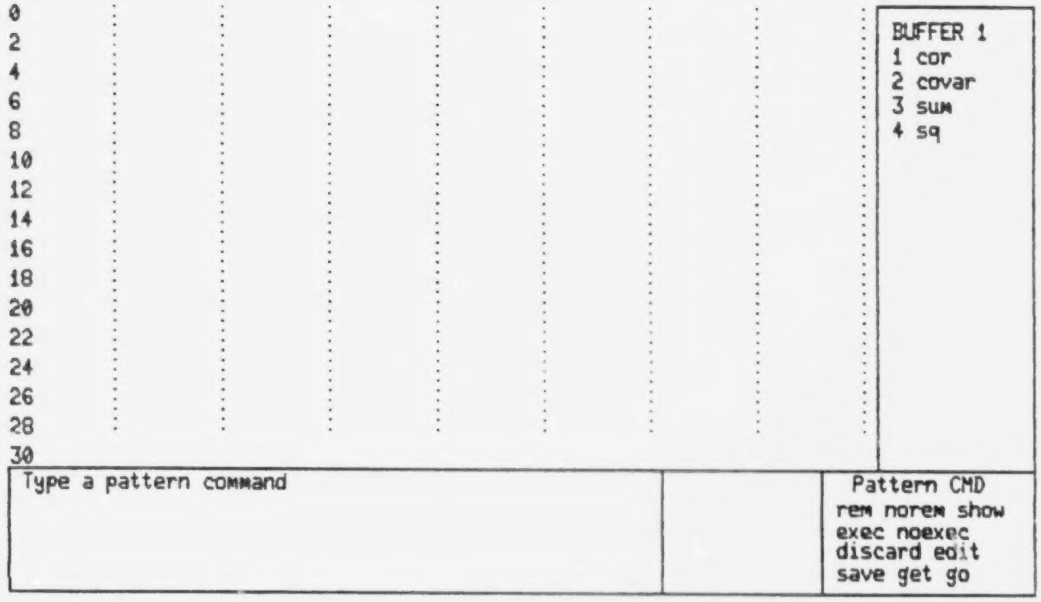

pattern
remember/noremember
execute/noexecute pattern
discard pattern
edit pattern
save pattern
get pattern
show
go
pattern
names
notes
replace line
delete line
insert line
show pattern
stop

This command is available in the main menu and under the "answer" option of the main menu. This option is provided so that an author can create a template based on the commands used to create a sequence of events or processes. For example, the author has to frequently define the same general algorithm for the identification of a answer analysis sequence, or for the presentation of a set of examination questions. The option allows the author to save a set of commands and have them repeated, with appropriate stops for input of text, as would be required with varying reply messages to the student. In addition a pattern may be named and saved, so that they may be used over again. The author is also able to edit a pattern by replacing a command, deleting a command, or inserting a command in the list of commands saved.

The use of pattern is initiated at the point in the authoring process when all further commands are to be saved. (Some commands, like "show", "help", "ser", "quit", "cmds", and the pattern commands themselves are not saved).

2.0 Pattern Options

When the option "pattern" is selected the following menu is displayed:



Type a pattern command

While this option is enabled certain input cannot be saved, eg., text being placed on the screen cannot be saved, as one would expect the text to vary with repeated use of the pattern. The effect of each pattern option is noted below:

rem for "remember"; this enables the pattern feature. Upon leaving the pattern mode and returning to the last menu, all commands from that point on will be saved. When "rem" is set, it will appear in red and blinking.

norem turns the pattern feature off. If "rem" had been set it will reset from blinking in red, to a yellow no-blinking mode.

show causes the list of saved pattern names or the currently constructed pattern of commands to be displayed for the author to see. The prompt is "Show pattern, names, or notes; stop".

a) **pattern** will cause the currently saved pattern to be displayed; If no pattern exists the word "empty" will appear in the display area.

b) **names** will cause the pattern names to be displayed; these names are global to the course

c) **notes** will ask for the pattern name for which the note is requested; then the note (comment about the pattern saved at the time the pattern was created) is displayed.

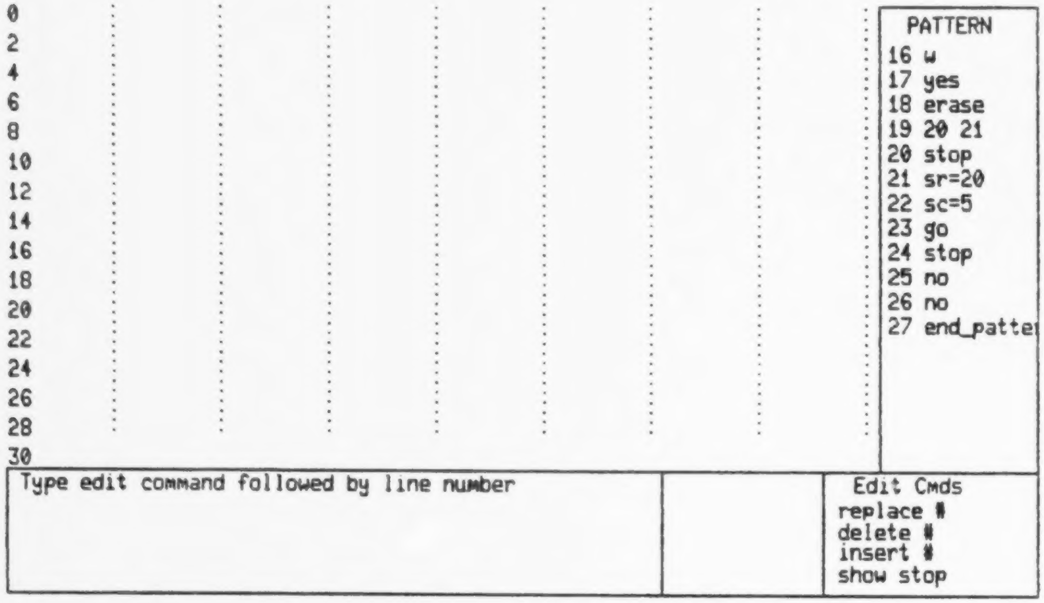
exec causes the automatic execution of the saved commands to take over and drive the authoring processes. "norem" is automatically turned off, as a pattern cannot be used to create another pattern.

noexec disables the automatic execution of the commands saved

discard throws away the existing pattern; a new pattern cannot be created unless the current one is "discard"ed.

edit allows editing of a current pattern: when called the following are available:

1) **replace #** is used to replace the command with the line number #
2) **delete #** is used to delete the command at line numbered #
3) **insert #** is used to insert a new command after line numbered #
4) **show** re-displays the current pattern
5) **stop** is used to exit from edit mode.



```
PATTERN
16 w
17 yes
18 erase
19 20 21
20 stop
21 sr=20
22 sc=5
23 go
24 stop
25 no
26 no
27 end_patte
```

```
Type edit command followed by line number
```

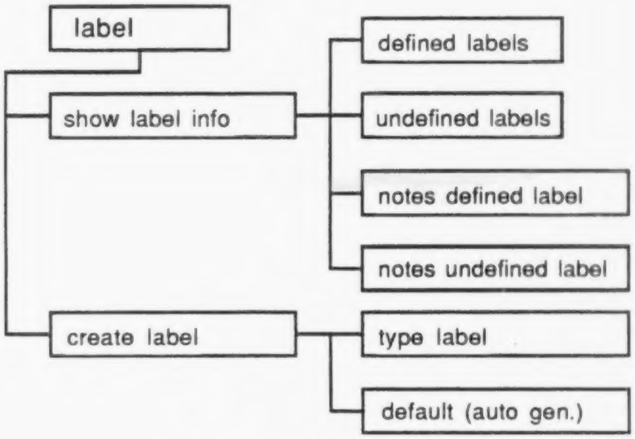
save allows the saving of the current pattern; the author is prompted by "Type pattern name; max 10 char", and then for a comment for the pattern. If a save has been successfully made, the message "pattern saved as:" followed by the name is displayed.

get allows the recall of a previously saved pattern. The author is prompted for the pattern name. If the pattern is found, the message "done and displayed" will appear, and the pattern is displayed to a maximum of 15 lines. If more than 15 lines are required, the message "Return to show more or stop" is displayed.

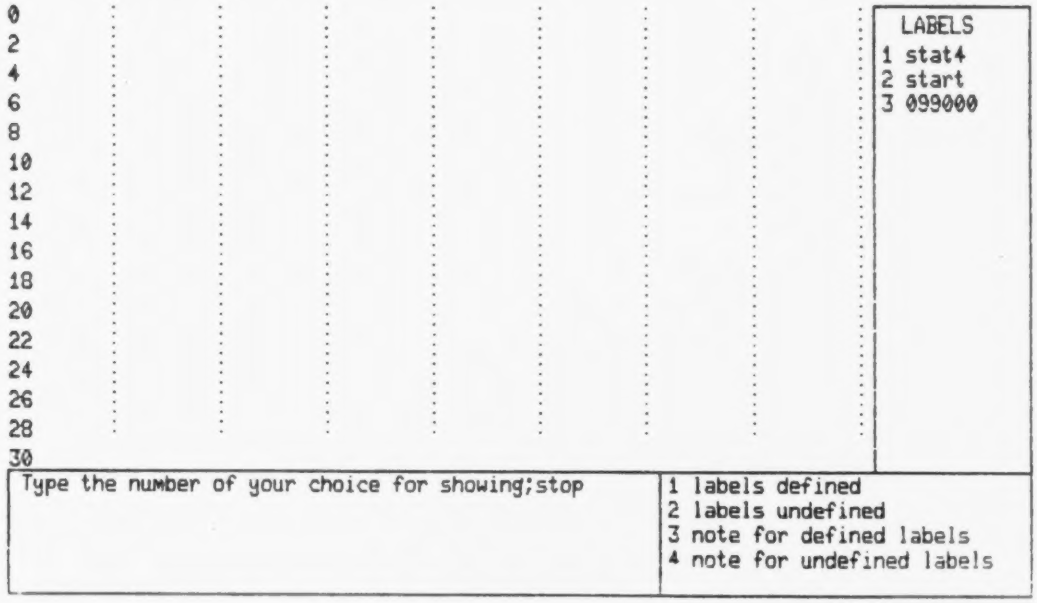
go returns to the calling menu; if "rem" was enabled, the commands from that point on will be saved until a return is made to the pattern menu and "norem" selected. Commands such as "help", "show", etc. are not saved. If "exec" was enabled, the commands from the current pattern will be taken and executed, with keyboard input being requested at appropriate points.

Section 9
LABEL

1.0 Introduction


label
show label info
create label
defined labels
undefined labels
notes defined label
notes undefined label
type label
default (auto gen.)

This feature allows the author to define an entry point into the course code to which an explicit branch may be made during course execution. Upon selecting this option, the author is queried as to whether he wishes to "show", "create", "help", or "stop". For a course authored for the first time, the authoring system automatically creates 3 labels. These labels are 1) a label identical to the course name, 2) the label "start", and 3) a numeric label made from the segment number and 000. These labels cannot be created by the author. These labels should not be changed by the author as they are used under default conditions by the CAI run time system.



```
0                                                         LABELS
2                                                         1 stat4
4                                                         2 start
6                                                         3 099000
8
10
12
14
16
18
20
22
24
26
28
30
Type the number of your choice for showing;stop    1 labels defined
                                                   2 labels undefined
                                                   3 note for defined labels
                                                   4 note for undefined labels
```

2.0 Label Options

The following options are available to the author:

show: show provides entry into a procedure to display the labels and associated notes created in the current session, as well as those created in previous authoring session. The author is allowed by choice of options the type of information to be displayed. The menu displayed contains the following items:

a) 1 labels defined
b) 2 labels undefined
c) 3 note for defined labels
d) 4 note for undefined labels

The reply to the menu is typing a digit in the range 1 to 4. "labels undefined" refer to those labels to which a branch has been made but which have yet to be defined by the author. (Such labels are listed as P-Labels on entry to the authoring system). The author may view the list of labels created and their associated notes, or the list of labels to which a branch has been defined, but for which the exact branch-to point has not been defined. The note associated with each label is requested of the author each time a "create" is selected, i.e, when a new label is created. The comment is optional, but an author is wise to comment a label if that label is a beginning of a content section to which a branch might later be made in the authoring of the course. Eventually, all undefined or pending labels must be resolved by defining the label of the same name.

create: If "create" is selected, the prompt "Type the label, or CR to default" is given. CR means to touch the RETURN key and is the default case. In the case of a default the last calculated default label, made up of the segment number (3 digits) followed by a 3 digit number which initially starts at 000 and is incremented by 004, is defined. A label typed by the author is restricted to 6 characters. In either case of defining the label, the author is queried for a note or comment. It is recommended that authors do not create many word-type labels as there is only a finite number of such words which can be created without the names becoming so similar that confusion results. The label, if generated or entered by the author, is echoed for the author to see.

When a label list from a previous authoring session is loaded, the list is searched from bottom to top until the last automatically generated label is found, it then forms the basis for the generation of other labels created by default for the current session.

In the case of a label entered by the author, the list of labels is searched to ensure that a duplicate label is not being entered.

When a new label is entered, a search is made of the list of undefined (pending) labels to determine whether the new label matches one of the undefined ones. If the new label does match an undefined label, the undefined label is removed from its list (as it is now defined).

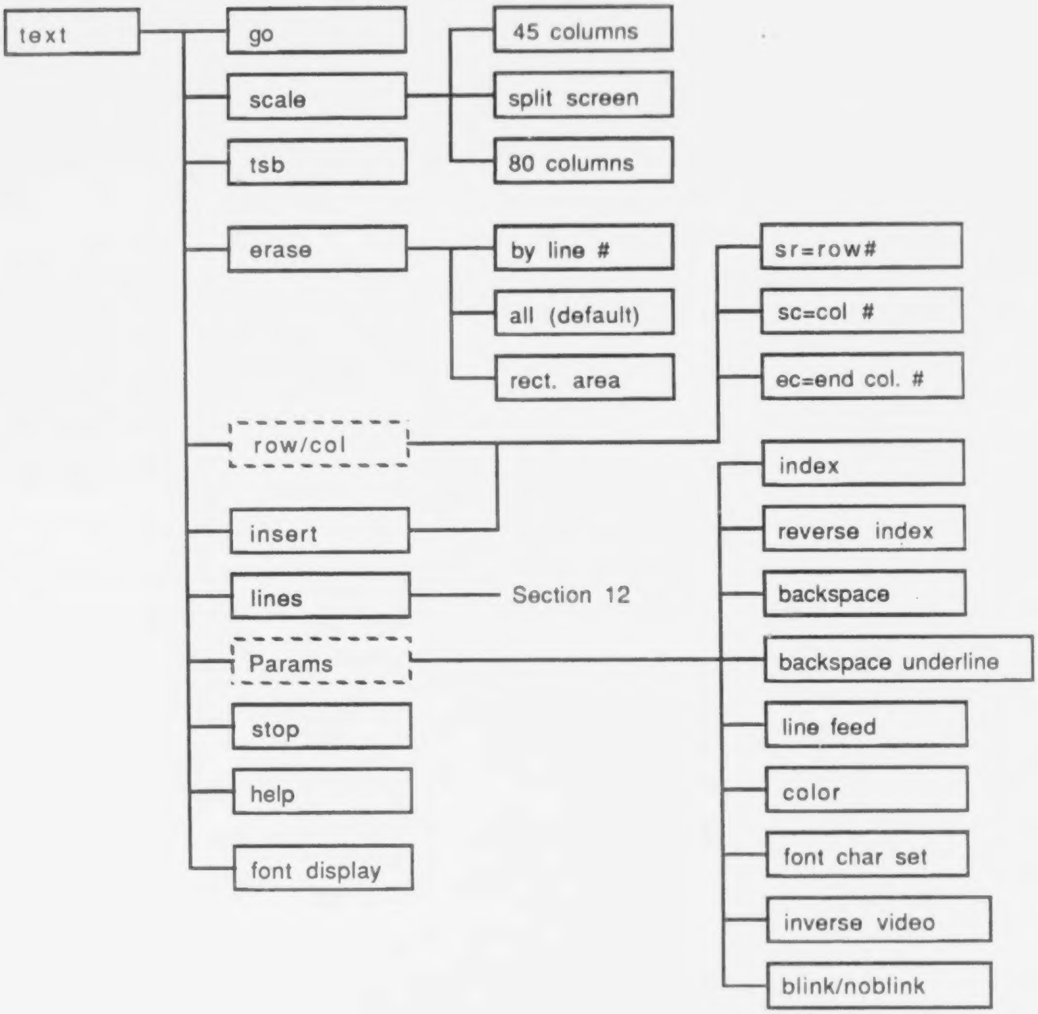
Each time a branch is defined (as may occur in the answer analysis procedure) the label is searched for in the defined label list. If it is present in the defined label list, the branch-to label is defined and nothing is done. If the branch-to label is not in the defined label list it is then added to the undefined list.

Section 10
pr prr:cntl

The pr and prr option provide points to which implicit branching may occur. An implicit branch is defined as a point to which a branch is automatically made without that point having a label. The pr may be considered a "new problem", while the prr may be considered a "new problem and restart point". A restart point is that point at which the student will be restarted after leaving the course and returning. Normally, it should be a point after which the screen format is set and the screen is erased.

An automatic branch is made to a pr or prr point after a correct answer has been identified, and further answer checking is not required. If a wrong answer or an unrecognized answer has been identified then encountering a pr or prr will cause a branch to be made to the last point at which the keyboard was opened for a response from the student. A pr is always created when the author leaves the "answer" option of the main menu.

Section 11
TEXT COMMANDS


text
go
scale
tsb
erase
row/col
insert
lines
Params
stop
help
font display
45 columns
split screen
80 columns
by line #
all (default)
rect. area
Section 12
sr=row#
sc=col #
ec=end col. #
index
reverse index
backspace
backspace underline
line feed
color
font char set
inverse video
blink/noblink

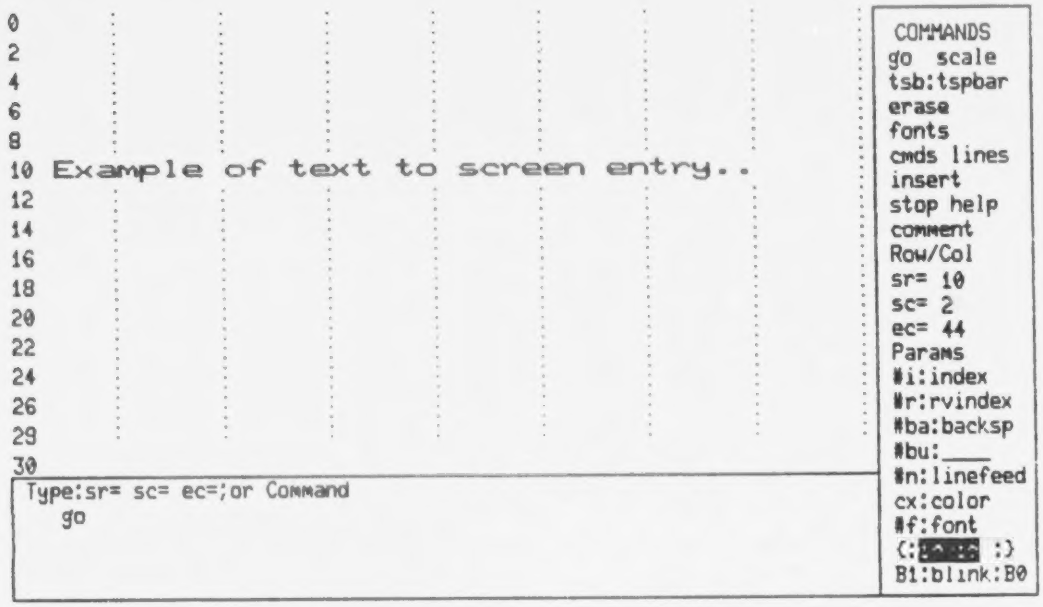
The Commands are divisible into three types:

1) those which are auxilliary to placing text on the screen, e.g., erase, fonts,

2) parameters for setting the cursor's initial position on the screen, and

3) attributes (e.g., color) of the characters being placed to the screen, and special cursor control such as required for superscripts and subscripts.

The display shown below is presented when the text option is taken at the main menu level.



```
10 Example of text to screen entry..
```

```
Type:sr= sc= ec=;or Command
   go
```

0
2
4
6
8
10
12
14
16
18
20
22
24
26
29
30
COMMANDS
go scale
tsb:tspbar
erase
fonts
cmds lines
insert
stop help
comment
Row/Col
sr= 10
sc= 2
ec= 44
Params
#i:index
#r:rvindex
#ba:backsp
#bu:____
#n:linefeed
cx:color
#f:font
{:[illegible] :}
B1:blink:B0

1.0 General Sequence

A typical sequence for placing text to the screen would be as follows:

1) at the main menu level, a screen erasure is carried out, or, it may be done at the Text menu level.

2) the screen is scaled using the scale command

3) the parameters for positioning the cursor to its initial screen position (sr=, sc=, ec=) are set, and the command "go" is given.

4) the cursor is then automatically positioned and the author may start typing the text. RETURN is used to start a new line during screen entry. Note that if the current line contains subscripts, then the RETURN should not be used since this will position the cursor 2 rows lower, while 3 rows are required. To obtain a 3 row advance use the #n option with #=3, eg., 3n .

5) attributes and special functions may then be called for by breaking out of the screen formatting mode by using the ESC key, and the commands listed in the menu under the term PARAMS may be used.

6) the command "tsb" is used to hold the screen for the student to read.

7) an erasure of the previous screen, or portion thereof is made using the erase command, and followed by setting a new initial cursor position to place more text on the screen. (The scale and cmds commands may be used to re-establish the scaling lines and command box).

NOTE OF CAUTION:

When using the DEC Gigi terminal:

a) DO NOT USE THE BACKSPACE KEY TO ERASE INCORRECT CHARACTERS IN THE CURRENT LINE; USE THE DELETE KEY.

b) DO NOT USE THE BACKSPACE KEY FOR PURPOSE OF UNDERLINING; USE THE "#ba" option for backspacing, and the "#bu" for backspacing and underlining, where # is the number of columns to backspace.

c) DO NOT USE THE BREAK KEY UNDER ANY CIRCUMSTANCES.

d) Text may be corrected only for the current line using the DELETE key up to the point, but not beyond, the point where the ESC key may have been used.

2.0 TEXT Commands and Parameters

These commands include go, scale, tspbar, erase, fonts, cmds, lines, insert, stop and help.

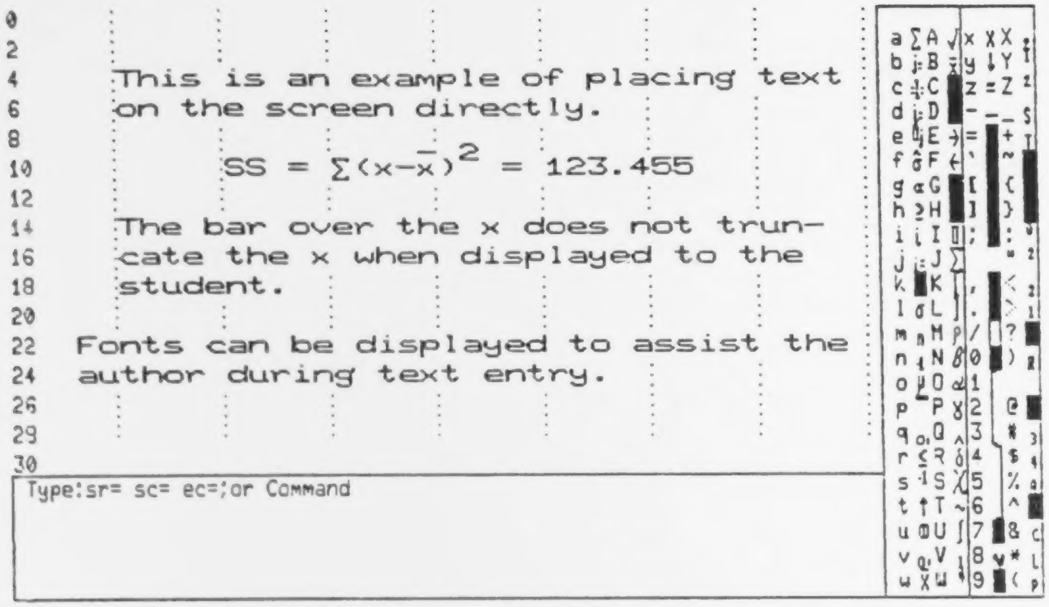
go is used after the definition of the initial cursor position parameters have been made. It causes the cursor to be placed on the screen for entry of text.

scale is used to rescale the screen and is automatically done after an erase.

tspbar causes a 'touch space bar' on the screen in row 30 to hold the screen for the student to read.

erase is used to erase the whole screen, a selected number of rows, or a selected area of the screen.

fonts allows the author to call up and have displayed an alternate set of characters fonts. This display is made on the right-hand side of the screen. This display allows the author to select which character defined on the keyboard will produce the new desired character. An example of this display follows.



```
0
2
4      This is an example of placing text
6      on the screen directly.
8
10            SS = Σ(x-x̄)² = 123.455
12
14     The bar over the x does not trun-
16     cate the x when displayed to the
18     student.
20
22   Fonts can be displayed to assist the
24   author during text entry.
26
29
30
Type:sr= sc= ec=;or Command
```

cmds is used to re-write the box containing the commands.

lines is used to provide a facility for drawing lines, boxes, and circles during the text to screen operation, e.g., creating a pointer, a box around text etc.

insert command will allow display of a numeric variable which has been created by the arith command option in the main menu or a string of text loaded in a buffer (see store option in the main menu). The name of the variable will appear on the screen in blink mode. The author is questioned as to the row and column where the insert is to be made.

stop causes a return to the last menu.

help provides guidance in the use of the commands.

3.0 ROW/COL

Under the ROW/COL, the term

sr refers to starting row,
sc to starting column, and
ec to ending column.

The value 0 is the default value. sr=0 indicates the starting row is set to zero, sc=0 indicates the starting column is set to zero, and ec=45 indicates that the ending column is set to 45. The ending column (ec) is defaulted to 40,45, or 80 depending upon the screen selected. The author may redefine these to any legitimate value. For example, to reset the starting row to 10, the author would type sr=10. The new value is indicated in blink mode in the box. The values of sc and ec are set in the same manner. Once these are set to the desired values, typing "go" places the cursor to the defined row-col position for text entry, and the author may begin to enter text to the screen. The RETURN key is used to start a new line, BUT THE ESC KEY IS USED TO BREAK OUT OF TEXT ENTRY ON THE SCREEN. The use of the ESC key allows the author to change attributes such as color, to carry out special functions such as re-positioning the cursor to get subscripts, or to stop the entry process. The commands which become available to the author after depressing the ESC key are listed in the menu under the term "Params", as well as the command word "stop".

If text, or a drawing, exists to the right of the text currently being placed on the screen, then the ec (ending column) parameter must be set, otherwise the current text will cause an erasure to the last available column.

Normally, the RETURN key is used to move the cursor on the screen to the beginning of a new line. However, the parameter #n will accomplish the same effect when # is 2, ie., 2n. If the current line uses subscripts which extend one row lower than the current line, one should use 3n to take the cursor directly to the next line far enough down so that the subscripts of the current line are not partially erased by the new line.

It is recommended that the author not attempt to place text into the full width of the available screen, e.g., use 40 of the 45 columns available. The additional 5 columns not used become very useful when additional characters have to be added to the line in the editing mode.

When major errors are made in a line, the simplest procedure to correct these is to touch ESC to break out of the screen mode, and type the command stop. Select the comment option and type a comment which will identify the nature of the error in the course code for later correction using the editor. Then proceed to reset the row and column parameters and re-enter the text correctly.

4.0 Params

Access to the functions listed under "Params" is made by using the ESC key while the cursor is positioned on the screen for entry of text. The functions available are as follows:

stop: causes the entry of data to the screen to be stopped. sr, sc and ec are reset to their default values, and the process of positioning the cursor for text entry may start again. A second "stop" is required to leave the text entry mode and to return to the last menu.

#i: index - use #i where # is a number to indicate the number of index positions the cursor should be moved down; used for subscripting. For example, "1i" followed by a RETURN will replace the cursor on the screen 1 row lower than before (2 rows = 1 line of text).

#r: rvindex - use #r. #r operates in the same manner as #i but the cursor is repositioned # row(s) higher than the position from which it left; used for superscripts

#ba: backsp - use #ba. #ba provides back spacing of the cursor # column positions.

#bu:______: backspaces # column positions and then underlines to the original position of the cursor.

#n: linefeed provides a return of the cursor to the starting column and 1 row lower (1 row lower is equivalent to one-half a line of text).

cx: color - use cx where x is the first letter of one of the colors green, cyan, magenta, red, blue, white, or yellow.

#f:font - use #f where # represents the number of the alternate font set to be used while entering text to the screen. The author must have loaded the required font to the Gigi terminal before starting an authoring session. (This will be changed so that it can be done when required). The following example would place the Greek summation character on the screen: <ESC>1f<CR>a<ESC>0f <CR>. "1f" sets font 1, "a" is the keyboard character equivalent to the sum symbol, "0f" resets the font to set 0.

{ sets the text into inverse video.

} resets the inverse video to normal mode.

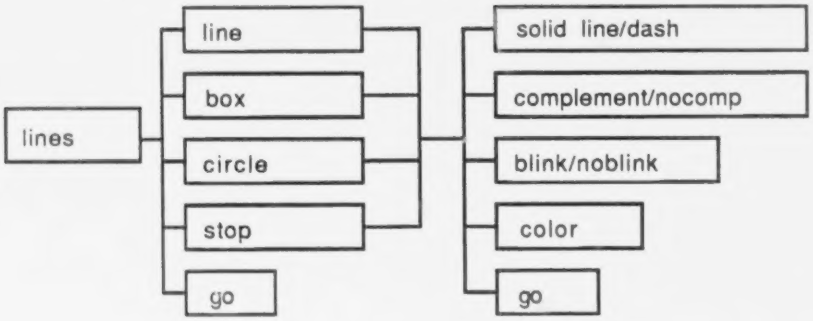
B1 turns blinking on (B must be uppercase).

B0 turns blinking off (B must be uppercase).

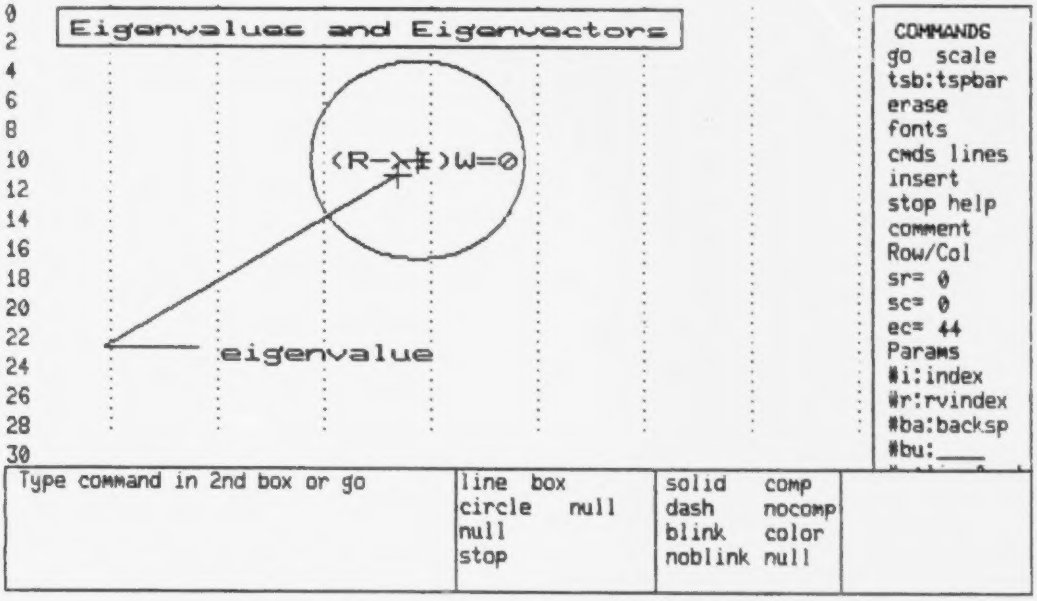
NOTE: After typing the parameter, and upon depressing the RETURN key, the cursor will return to its screen position before the ESC key was used. From this point onward, the new attribute is in effect.

Section 12
Lines

1.0 Introduction


lines
line
box
circle
stop
go
solid line/dash
complement/nocomp
blink/noblink
color
go

This option can be used by the author to create simple geometric figures. For example, an author may wish to draw a box around a title or a paragraph.


0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
Eigenvalues and Eigenvectors
(R-λI)W=0
eigenvalue
COMMANDS
go scale
tsb:tspbar
erase
fonts
cmds lines
insert
stop help
comment
Row/Col
sr= 0
sc= 0
ec= 44
Params
#i:index
#r:rvindex
#ba:backsp
#bu:____
line box
circle null
null
stop
solid comp
dash nocomp
blink color
noblink null

Type command in 2nd box or go

2.0 Options

A menu is displayed containing the following commands:

line: used to draw line segments using the cross hairs

box: used to draw a rectangle by marking the upper left and lower right corner of the box to be drawn

circle: to draw a circle; the center and a point on the circumference are marked using the cross hairs

stop: to stop the drawing process position the cross hairs over the word stop and touch the return key

null: not used

Once a selection is made the following attributes may be set by typing the word:

solid or a **dashed** line: If dashed is selected a third menu displaying 4 different patterns for the dashed line is presented, and the author prompted for a choice.

blink or **noblink**

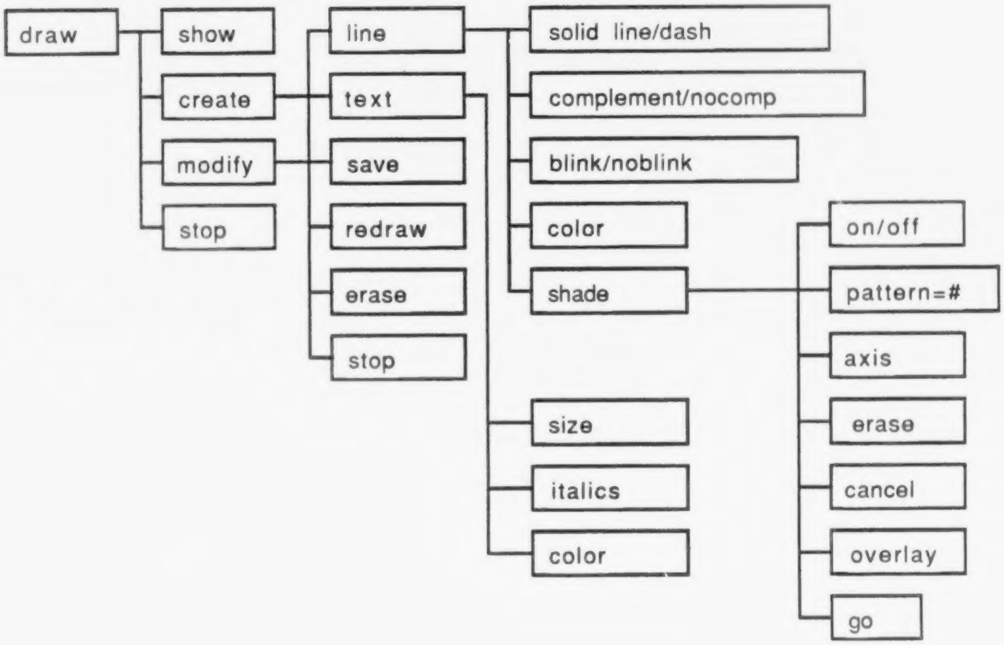
complement or **nocomplement**; when a line passes through a shaded plane, use complement.

color: the author is prompted for the color and the option "color" is set to the color selected.

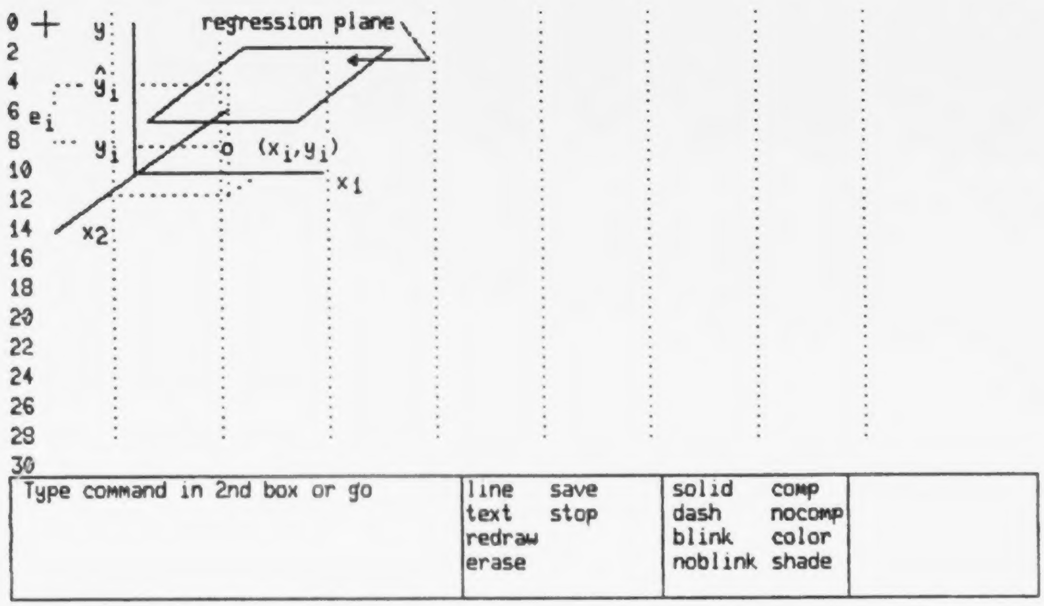
null is not used

The author can proceed to draw by typing "go". The cross hairs will automatically appear on the screen. The author positions the cross hairs to the desired location by using the arrow keys, and touches the RETURN key to mark the first and successive points.

Section 13
Draw


draw
show
create
modify
stop
line
text
save
redraw
erase
stop
solid line/dash
complement/nocomp
blink/noblink
color
shade
size
italics
color
on/off
pattern=#
axis
erase
cancel
overlay
go

The draw mode is entered from the main menu. This mode is used to create either drw (relocatable) or fp (fixed position for the split screen format) type drawings.


0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
y
ŷi
ei
yi
regression plane
o (xi,yi)
x1
x2
line save
text stop
redraw
erase
solid comp
dash nocomp
blink color
noblink shade

Type command in 2nd box or go

1.0 Drawing Commands

The author should first create a drawing on the screen and then build the text around it. Placing text on the screen and then a drawing would require the author to know the exact area occupied by the drawing to prevent the drawing over writing the text.

Upon entry to the drawing mode, the author is queried as to the type of action desired. These options are as follows:

show to show the list of names and comments associated with drawings which already exist,

create to create a new drawing,

modify to modify (add to) an existing drawing, and

stop to return to the main menu.

If "show" is selected the author is asked whether he wishes to see the list of picture "ids" or the "notes" associated with a given id.

If "create" is selected, the author is asked to define the type of drawing (drw or fp) to be created. The author is then queried as to the 3 character id if the drawing is a drw type, or a 3 digit id if the drawing is of a fp type. The author is then asked to use the cross hairs and mark an absolute reference point for the drawing. This point may be considered to be the upper left-hand corner of the area to be occupied by the drawing being created.This point will be the point by which the drawing will later be positioned during the course execution.

Two menus arranged at the bottom of the screen are used to provide options to the author (see previous example). The following interpretation is given to the commands displayed in white in the first menu:

line: provides for line drawing (See below regarding attributes for line drawing)

text: provides for positioning the cursor to the screen for purposes of labeling components of the drawing created.

redraw: carries out a re-draw of the drawing, displaying it without all the cross hair target marks. These targets marks are placed on the screen when a point is marked to guide the drawing process. "redraw" re-displays the drawing as the student would see it. The author may continue to add to the drawing.

erase: erase an area of the screen starting with the marked position about which absolute addresses are calculated, to a lower right position. The author is asked to define the lower right position of the erase area.

save: provides for creation of the course code necessary to display the drawing. This option was made available because an author may wish to make a drawing, which creates the appropriate Regis file, without necessarily creating the code for display of the drawing in a course. (Regis is the DEC graphics code used by the Gigi terminal).

stop: causes a return to the last menu. If the cross hairs are being used, they must be positioned over the word stop and the RETURN key depressed.

2.0 Line Drawing Attributes

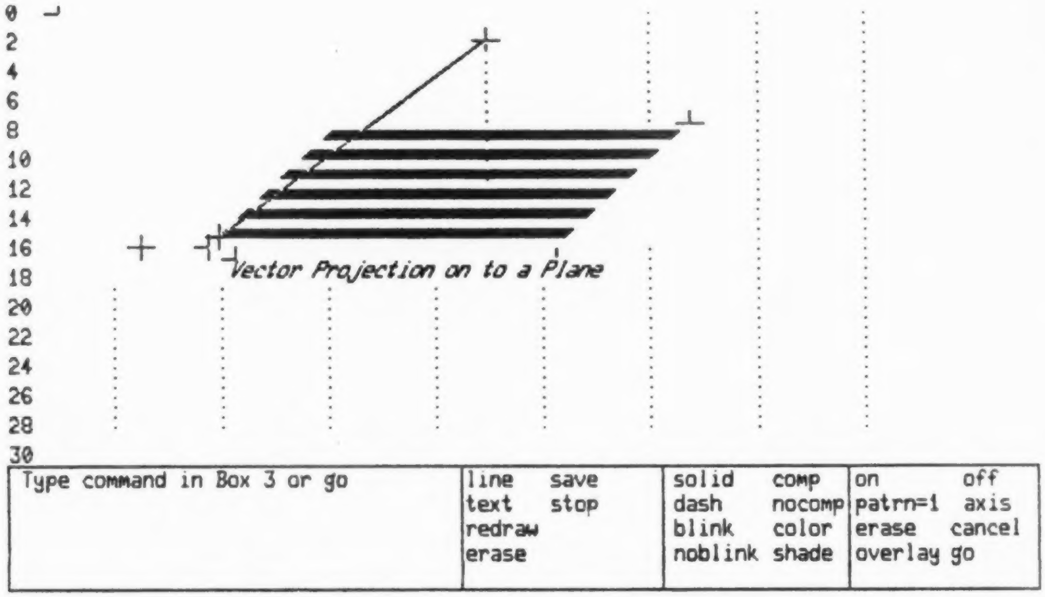
Once the line option has been selected, the second box contains a new set of attributes which can be set for line drawing. As each attribute is set its color is changed. In the case of the color attribute, its color is changed to the color selected.

solid:	indicates a solid line is required; solid is the default value.
dash:	indicates that a dashed line is required.
blink:	indicates that "blink" has been enabled to cause the line from this point on to blink.
noblink:	disables the blink attribute.
comp:	sets complement writing.
nocomp:	disables complement writing.
color:	allows setting of the color from the current draw position for all subsequent moves; the word color changes to the color selected.
shade:	allows the definition of the y-axis to which shading will be made as lines are drawn. (This option allows for a "fill" procedure, but under Regis restrictions it is an obtuse procedure which is procedurally difficult to use). A new box (a third box) of commands is displayed. (See below).

3.0 Marking Points on the Screen

During the line drawing process, the author may be asked to mark a point on the screen with cross hairs. The author must move the cross hairs to the desired location on the screen and touch the RETURN key to register the point. The cross hairs are moved about the screen using the arrows keys. In order to speed up the movement of the cross hairs on the screen hold down the SHIFT key while holding down one of the arrow keys. To stop the draw process while using the cross hairs, place the cross hairs over the word stop and touch the RETURN key.

4.0 Shading to the Y-axis



```
Type command in Box 3 or go   | line   save | solid   comp   | on      off
                              | text   stop | dash    nocomp | patrn=1 axis
                              | redraw      | blink   color  | erase   cancel
                              | erase       | noblink shade  | overlay go
```

When "shading" is enabled, each time a line segment is drawn the area between the line segment and a predefined y-axis location is colored solid, or by option, a pattern. When the shade option is selected, a third menu is displayed with the following:

on and **off:** enable and disable shading.

patrn=1: allows the setting of shading patterns 1,2,3 or 4 as defined by Regis.

erase: equivalent to shading in the color "black" for cutting off a shaded area

overlay: provides for overlay writing mode.

axis: allows the definition of the y-axis to which shading will be done as a line segment is drawn

cancel: provides for cancelling of all attributes associated with shading.

go: indicates a return to the last set of commands.

Note: The shading feature has not been used very much and therefore is likely to have bugs or inadequate author guiding features. A typical sequence to create a shaded area would involve the following steps:

1) select "line" and the attributes desired

2) select "go"

3) mark the first point

4) select "shade", "on", "axis" and mark the y-axis to which shading is to be made.

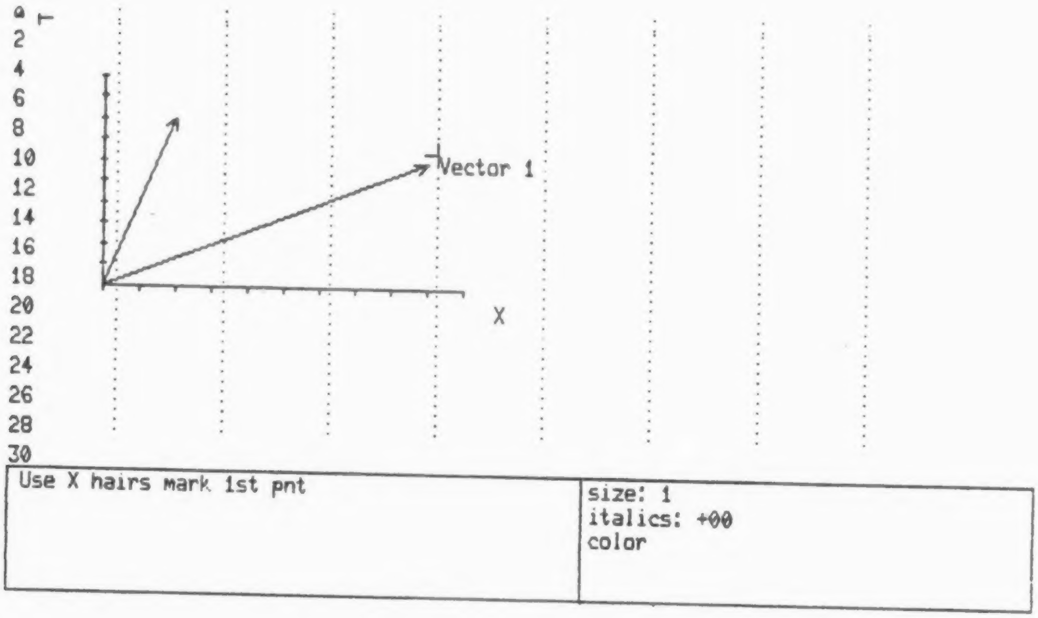
5) select "go" and using the cross hairs mark the second point.

6) remember to set shading off (cancel) when the feature is no longer needed.

It should be noted that in drawing a shaded parallelogram, the "erase" feature will have to be used to erase the shading to the y-axis when the last point drawn extends beyond the lower corner of the figure.

5.0 Labelling a Drawing

The text option allows the author to position the cursor for purposes of placing labels on a diagram. At the moment, this feature does not call upon the normal TEXT option for writing to the screen and therefore does not have all the features of writing subscripts, superscripts, and changing character fonts. This is an inadequacy which will be corrected. The following attributes may be set under text mode:



Use X hairs mark 1st pnt

size: 1
italics: +00
color

size 1 is set by default. The author may set size to a value between 0 and 16, however, size 0 is frequently too small and hard to read, while sizes 3 or larger are too big. Type "size" to make the change. The author is then queried for the size parameter.

italics allows entry of an angle for slanting the italics. Experience suggests -30 or +30 (degrees) provides an acceptable slant. Type "italics" to change the slant. The author is then queried for the slant angle.

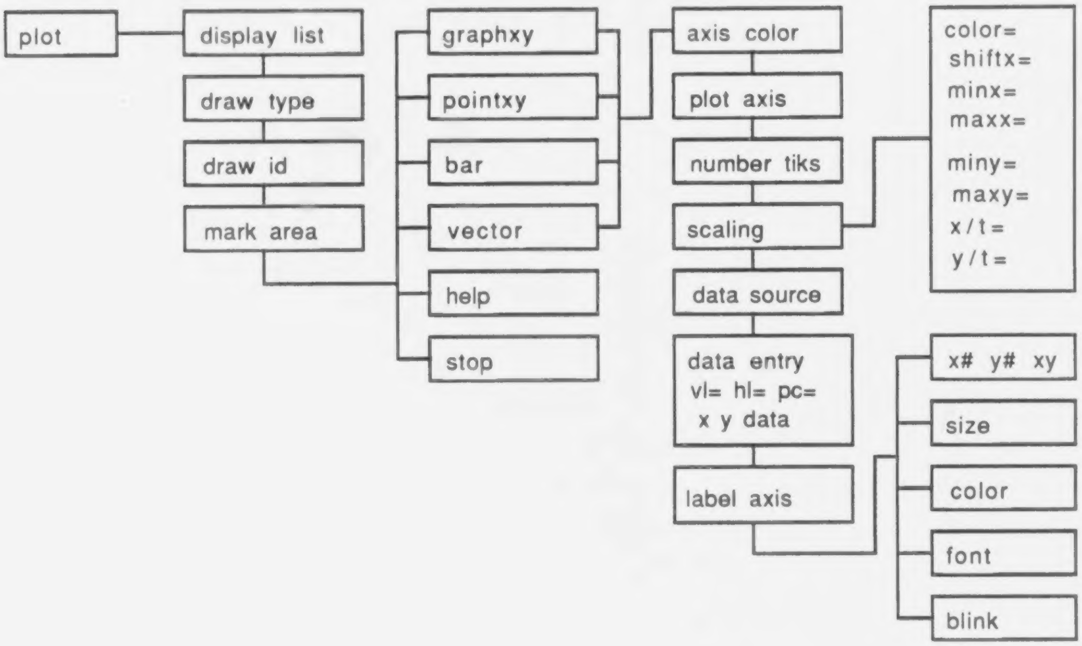
color may be set by typing "color". The author is then queried for the color. The word color is rewritten to the current color set by the author.

Note 1: To leave the screen text entry mode after placing text on the drawing, use the RETURN key. (This implies that only one line of text can be entered at a time.)

Note 2: The author need only be concerned with the drw-type drawing since it may be displayed under any screen format and erased using the erase option.

Section 14
Plot

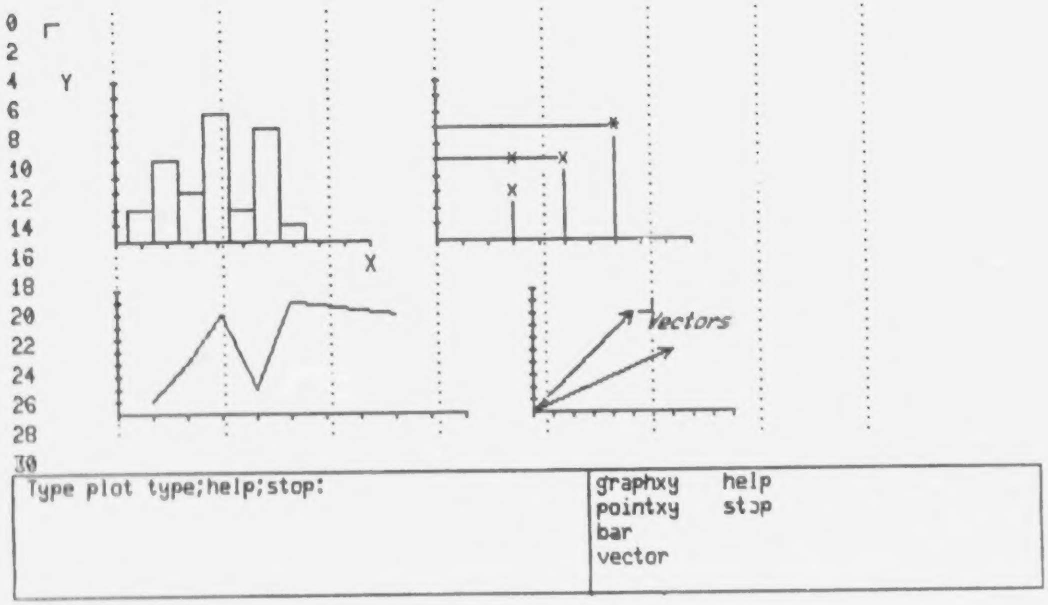
1.0 Bar, Point, Continuous, and Vector Plots


plot
display list
draw type
draw id
mark area
graphxy
pointxy
bar
vector
help
stop
axis color
plot axis
number tiks
scaling
data source
data entry
vl= hl= pc=
x y data
label axis
color=
shiftx=
minx=
maxx=
miny=
maxy=
x/t=
y/t=
x# y# xy
size
color
font
blink

The plot option allows the author to plot data in the following formats:

1) bar graph or histogram
2) point as in a scatterdiagram
3) continuous line function
4) vectors

An example follows illustrating the types of instructional plots which can be made:



```
Type plot type;help;stop:
```

Plots are named and stored in the same manner as described under the "draw" option. Data may be entered from the keyboard or from an existing data file. Four separate processes have been defined within the plot option. These processes are:

1) defining preliminary information about the drawing
2) drawing the axis
3) obtaining the plotting information for the axis drawn
4) making the plot
5) placing labels on the plot

2.0 Plot Prompt Sequence

2.1 Preliminary Information

The following prompt sequence is used:

1) **"Display current picture list? y or n."** The author may wish to review the current list of pictures (drawings of the drw-type or fp-type in order to prevent the creation of a duplicate picture name).

2) **"Is drawing type drw or fp?;stop"**. (A drw-type drawing is relocatable anywhere on the screen; an fp-type is positioned only on the right side of the split screen format.

3) **"Type 3 character identifier"**, followed by a request for a note about the drawing.

4) **"Mark upper left of plot area"** is requested to identify the point by which the drawing will be positioned.

5) **"Type plot type; help;stop"** and the following menu will appear:

graphxy	plots continuous line
pointxy	plots points using a character
bar	plots a bar graph
vector	plots vectors
help	
stop	

The author is then prompted for the axis color.

2.2 Drawing and Marking the Axes

The author is guided to draw the Y and X axis by using the cross hairs. The axes are subject to the constraint that they must touch or cross each other. If the condition is not met, a message is given and the process started again.

1)The author defines the number of "tik" marks to be placed on the y and x axis. These define the basic unit of each axis.

2)the author is then presented with a menu to define the scaling information.

2.3 Scaling Information

The author is required to define scaling information. The author may have the system determine the scaling information through knowledge of the minimum and maximum values of the x and y data. Alternatively, and usually preferably, specifying the value of each 'tik' unit on the X and Y axis can be made. The menu presented has the following options:

color=	plot color e.g., color=red
shiftx=0	add constant to x data
minx=0	set minimum x data value
maxx=0	set maximum x data value
miny=0	set minimum y data value
maxy=0	set maximum y data value
x/t=0	set value of 1 tik mark on X axis, eg., x/t=1
y/t=0	set value of 1 tik mark on Y axis, eg., y/t=2

The author may select to define the minimum and maximum values for x and y, or to define the value of each unit "tikked" on each axis. If the minimum and maximum values are set, then x/t and y/t need not be set. If x/t and y/t are set, then the minimum and maximum values are not set.

It is more useful for instructional graphics to define x/t and y/t as this ensures easy entry of integer data for the plot from the keyboard. (Under the labelling option, the author can label "tik" marks with any number or character which makes sense in the context of the instruction).

The author defines the source of input, i.e., from the keyboard or from a file

3.0 Entry of Data

The author then enters the data as x y points (a space must be placed between the two integers); "end" is to terminate data entry.

In the case of a pointxy plot, the additional features of

vl={on,off},
hl={on,off}, and
pc=x

appear as a menu. "vl" will cause a vertical line to be drawn from the point to the X axis, "hl" will cause a horizontal line to be drawn from the point to the Y axis. Setting hl or vl to on is followed by the prompt for the color of the line. Only after the color is set, is the term displayed as "on" in the color selected. The options "vl" and "hl" are set to off after the entry of a data point.

The author may wish to select such characters as "o,x,*,a,b" etc. "pc=x" (plot character equals x) indicates the point plotted will be with the character "x".

The data point entered is plotted immediately.

4.0 Labelling the Axes

The author is then given the option to label the plot. The menu provided is as follows:

x#={digit or character}; # is the tik position along the x axis which is to have below it the digit or characters. For example, x5=A would place "A" under the 5th tik on the x axis.

y#={digit or character}; interpreted same as x# described above but for y axis

xy will cause the cross hairs to appear; the cross hairs are positioned to where text is to be entered; pressing return will cause a cursor to appear for entry of the text

size=1; this allows specification of the size of text; usually 1,2, or 3 is adequate.

color=; allows specification of the color

font=0; allows specification of the font to be used

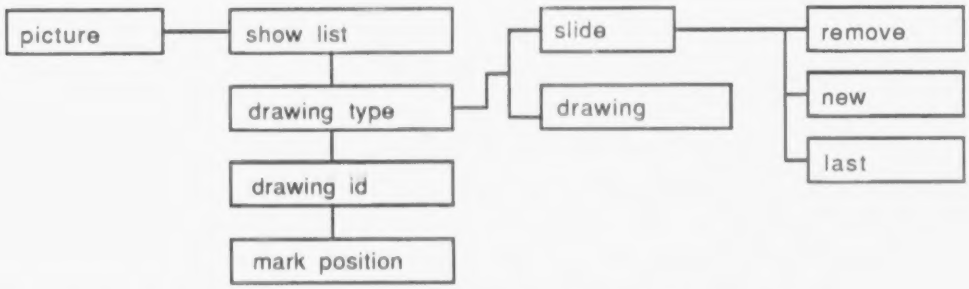
blink={off,on}; allows blink to be turned on or off

The author may then create a new plot using the previously defined axis, or a new axis.

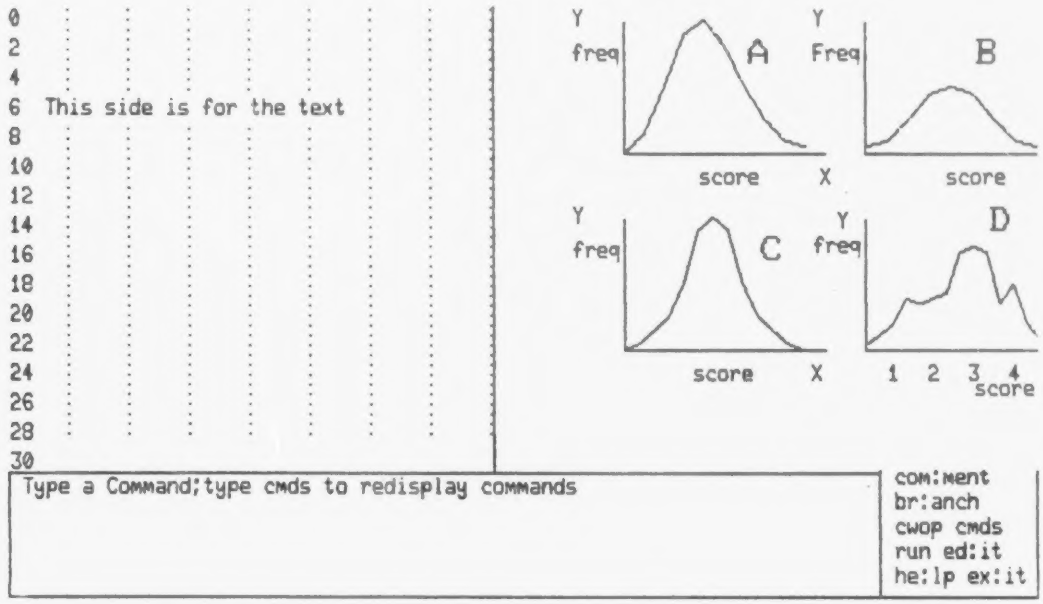
When multiple plots are made on different axes, they are treated as one unit for purposes of positioning during course execution. (All coordinates generated are in relative coordinate form).

Section 15
Picture

1.0 Introduction


picture
show list
drawing type
drawing id
mark position
slide
drawing
remove
new
last

An example of the the result of using the picture option is shown below.


0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
Y
freq
A
score
X
Y
Freq
B
score
Y
freq
C
score
X
Y
freq
D
1
2
3
4
score
com:ment
br:anch
cwop cmds
run ed:it
he:lp ex:it

This side is for the text

Type a Command;type cmds to redisplay commands

This option of the main menu allows for the recall and display of drawings of the drw and fp type which have already been created and exist as Regis files in the user's workspace. The author is asked to specify which type of drawing is being recalled. It must be noted that if an fp drawing is being recalled the "split" option under the "screen" option in the main menu must have been set in order to provide the split-screen mode. In the case of both types of drawings the author must know the identifier of the drawing, ie., must know the xxx part of drwxxx, and the nnn part of the fpnnn type drawing. fp-type drawings are automatically positioned on the right-side of the split screen, while for drw-type drawings, the author must mark, using the X hairs, the upper-left corner of the area in which the drawing is to be positioned. Normally, drawings should be positioned first, and then text placed around them.

2.0 Author Query

The author is first asked whether the list of drawing names is to be displayed. This is done so that the author can more easily recall the drawings already created.` The author is then queried as to whether he wishes to display a "drawing or slide". Drawing refers to the drw-type display, while slide to the fp-type display. In the case of a drw-type display, the author is then asked to give the 3-character identifier of the drawing. He is then asked to mark the position on the screen where the upper-left corner of the drawing should be started. (Presently, a search is made for the drawing in the author's current directory, and if not found a search will be made of the drawings used in stat2).

In the case the author wishes a "fp" type drawing, the following query is initiated: "new, last, or remove?". These have the following meaning (and are related to the previous use of a film projector):

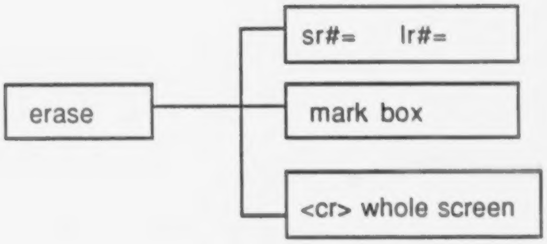
new - indicates that a fp-type drawing which is not the re-display of the last fp-type diagram is being called for. (Previously this meant that the film projector had to search and position the frame required, and then display it by opening the shutter).

last - indicates that the last presented picture is to be re-displayed. (Previously this meant that since a film frame had already been positioned, only the shutter had to be opened in order to display the picture).

remove - this has the effect of erasing the right side of the split-screen. (Previously, this operation was to close the shutter).

The author may call up a list of the drawings created and their associated notes.

Section 16
Erase



This option provides for the erasure of one or more lines of the screen for its full column width, to erase the complete screen, or to erase a defined rectangle of the screen. The author is prompted by "Type: sr#, lr#; mark;<cr>; help;stop".

sr# stands for the starting line number at which an erase is to start

lr# the last line number to which the erase is to be made. When the starting and ending line numbers are given the erase will be carried out over the full screen width for the screen format currently being used.

The sr and lr parameters are given together separated by a space, e.g., 5 10 would erase line 5 to 10 inclusive.

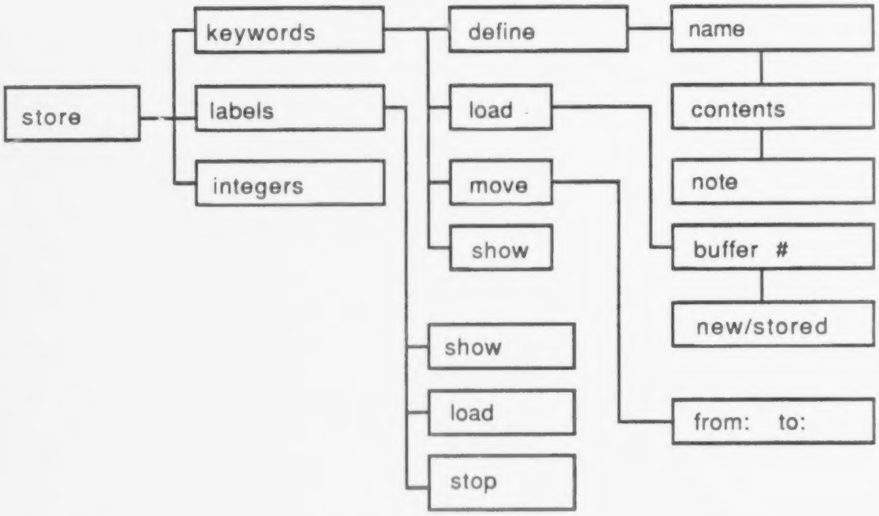
mark is used to erase any rectangular area of the screen. The cross hairs will be displayed, and the author is requested to use the cross hairs and mark the upper left and lower right of the rectangle to be erased.

<cr> is the notation for the RETURN key. This will cause the whole screen to be erased.

After either type of erase, the line numbers, the column guidelines, and the main menu is re-displayed.

The author should erase the complete screen when moving from a 45 column or 80 column format to a split screen format. Once in the split screen format, the erase option only erases the left side of the screen (where text is usually positioned). To erase the right side of the split screen, including an fp-type drawing, use the "close" option of the Picture command.

Section 17
STORE



1.0 Background

There are 2 types of buffers which they author may use. There are 6 buffers numbered 0 to 5 which are used only during course execution. These buffers usually contain such things as keywords used to identify a student's answer. For example, a buffer might hold "a b c d" for analyzing the response to a multiple choice question, or "mean mode median" to identify a response to a statistical question about central tendency. Buffer 0 is the buffer in which the student's answer appears, and therefore should not be loaded for use for such procedures as answer analysis where a string of keywords must be stored for later use.

The **"store"** command must be used to store a set of keywords if the Boolean answer analysis procedure (done by selecting the "answer" option in the main menu) is going to be used.

The author is also allowed to define and name an unlimited number of storage buffers for holding frequently used keyword strings, and which can be loaded to the buffers 1 to 5 for use during course execution.

Buffers may also be used for the generation of a reply to the student. For example, a part of the student's answer in buffer 0 might be appended to a string in buffer 1, and then buffer 1 displayed. (This facility is available through the "text" option).

2.0 The Store Options

Upon entering the store sequence, the author is asked whether keywords, integers or labels are to be stored. If "integers" is selected the author is asked to select the command "arith" in the main menu. NOTE: Digits may be stored in a buffer, but they cannot be manipulated arithmetically. If "keywords" is selected, the commands "define", "load", "move", "show", and "stop", are displayed.

define

Define allows the author to define the contents of a buffer and assign it a name; elements of a buffer are separated by one space. The prompt "Type name of buffer to be defined;8 char max" is given. The user then can type the name of his choice for the buffer. (Duplicate names are not allowed). The newly named buffer is added to the list of stored buffers and the list displayed. The user is then asked to "Type contents of the buffer", and then "Type note about the buffer". These buffers are saved between authoring sessions.

load

Load allows the author to load the contents of buffer 0 to 5 directly, or to place the contents of a named stored buffer into buffer 0 to 5. The author is prompted with "Type buffer index number (0-5) to be loaded", then "Load new or stored buffer?". If a stored buffer is to be loaded the author is prompted with "Type index number of stored buffer to be loaded" and a numbered list of stored buffer names is dispalyed. If the option "new" is taken, then the author must type in the specific contents of the buffer.

move

to move the contents of one buffer to another buffer. The author is asked the buffer number from which the contents are to be moved by the prompt "FROM:type buffer # 0-5" (this buffer still holds its original contents and in this sense the move is really a 'copy'). The author is then prompted with "TO:type buffer # 0-5" to define the buffer number to which the move is made.

show

Show is used to display the contents of a buffer which has been loaded or those which have been stored. The initial prompt is "Show loaded or stored buffer?". If "stored" is typed, then a numbered list of stored buffer names is displayed. Next, the author is prompted by "Type # to see contents" in order to see the contents of any one of the buffers. If the author asked to see a "loaded" buffer, he is prompted with "Type the number of the buffer", and upon typing the number (0-5) a numbered list, headed with "L-Buffer", of the buffer contents is displayed. "loaded" buffers are those which will be used during course execution.

stop

Stop is used to return to the last menu.

3.0 Loading Labels

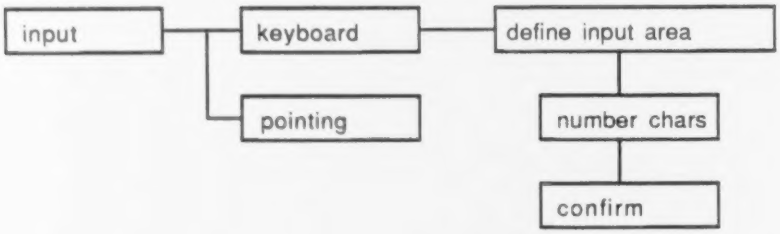
Labels may be loaded to "return registers" numbered 0 to 5. Branches may be made using these registers by branching to a numbered return register, e.g., br rr1, if rr1 contained the label 001099 then a branch to 001099 would be made during course execution.

If the choice is made to load "labels" a menu with the options "load", "show", and "stop" is displayed. If the option load is selected, the prompt "See label list?" is given in the case the author wishes to see labels which have been used previously and to which a branch might be made. Then, the prompt "Type label to be loaded" is given at which time the author is to enter the label to be stored. Then the author is prompted for a note about the label, and then requested to enter the register number in which the label is to be stored.

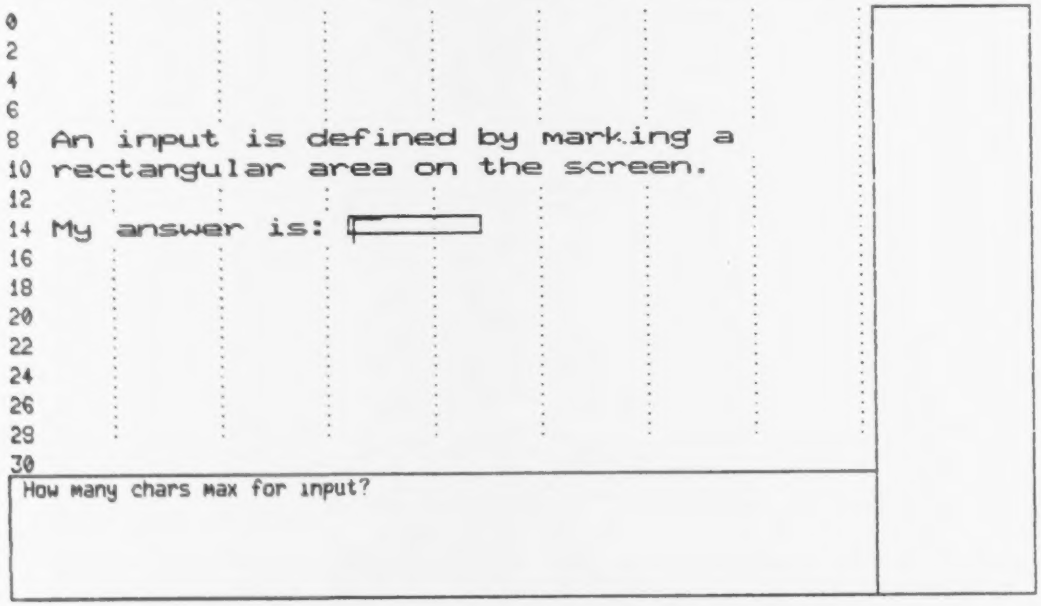
If the option "show" is taken, a display headed with the word "REGISTERS" is displayed, and the contents of each register shown, eg., rr0=001099, rr1=end, rr2=prob2.
When "stop" is selected, return is made to the main menu.

Section 18
INPUT

Introduction



The command "input" allows the author to indicate that an answer from the student is required. Both a keyboard and a pointing response may be defined by the author.



```
An input is defined by marking a
rectangular area on the screen.

My answer is:
```

```
How many chars max for input?
```

Upon selecting "input" the author is asked to specify whether a keyboard or pointing response is required. In the case of a keyboard response, the author is asked to use the cross hairs to define a rectangular area within which the student's response is to be made. The student's input will begin in the first row and column position of the area defined and extend for the number of input characters defined by the author. Only one line of input is allowed (this may be 40 characters for the split screen, 45 characters for the 45 column screen, or 80 characters for the 80 column screen.

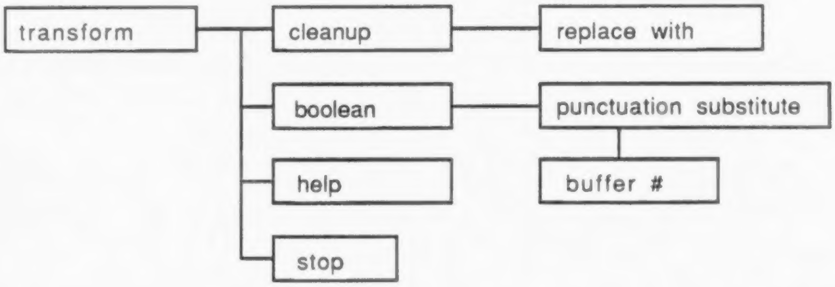
If the author defines a rectangle having a width less than one column, it is automatically set to 1 column; if the number of lines is less than 1, it is also automatically set to 1 line. If the order of the cross hair points is not upper left followed by lower right, the author is asked to remark the input area. If the author specifies a maximum number of characters of input more than the column width of the rectangle, it is automatically set to the width of the rectangle.

When the system time limit is exceeded during student input, the student will be returned to the point of the input.

Associated with each request for a response from the student is a 'response identifier'. This identifier forms part of the student's performance record to identify in the code exactly the origin of the input. The response identifier will be automatically generated to be the last label followed by a dash and the line number. The line number is the number of the line in the generated program code with the origin being the last label, e.g., 001003-45 would indicate the response was input at the 45th line under the label 001003.

Section 19
TRANSFRM

1.0 Introduction



The "transfrm" option is designed to provide the author with a set of transformations which may be placed upon the student's input response prior to the response being subject to an answer analysis. For example, the author may wish to remove all commas, colons, or numbers.

The option "transfrm" must be called after the "input" option and before a boolean answer analysis is called for through the "answer" option. It is not required for the use of the limit answer checking facility, i.e., when a numerical answer is checked for range.

Upon selecting this option the author is presented with the following options:

cleanup

"Cleanup" will generate the code to downshift the characters in the student's response, and to replace a defined string in the response with a string of the author's choice. The author is asked to define the replacer string and the string to be replaced. For example, a replacer string might be a space, and the string to be replaced might be a colon.

boolean

In the case of a Boolean analysis, the answer standard must be entered within (and). If the answer itself includes (and), as for example in (1,3), the cleanup function should be used to replace (and) with some other character.

The author is asked whether punctuation is to be replaced with blanks (a lowercase shift is done regardless). Then the author is asked for the buffer number containing the key words (see "store" option). The transform replaces the input string with a number corresponding to one or more of the stored keywords. The transformed string is then in the appropriate form for the boolean answer analysis.

help

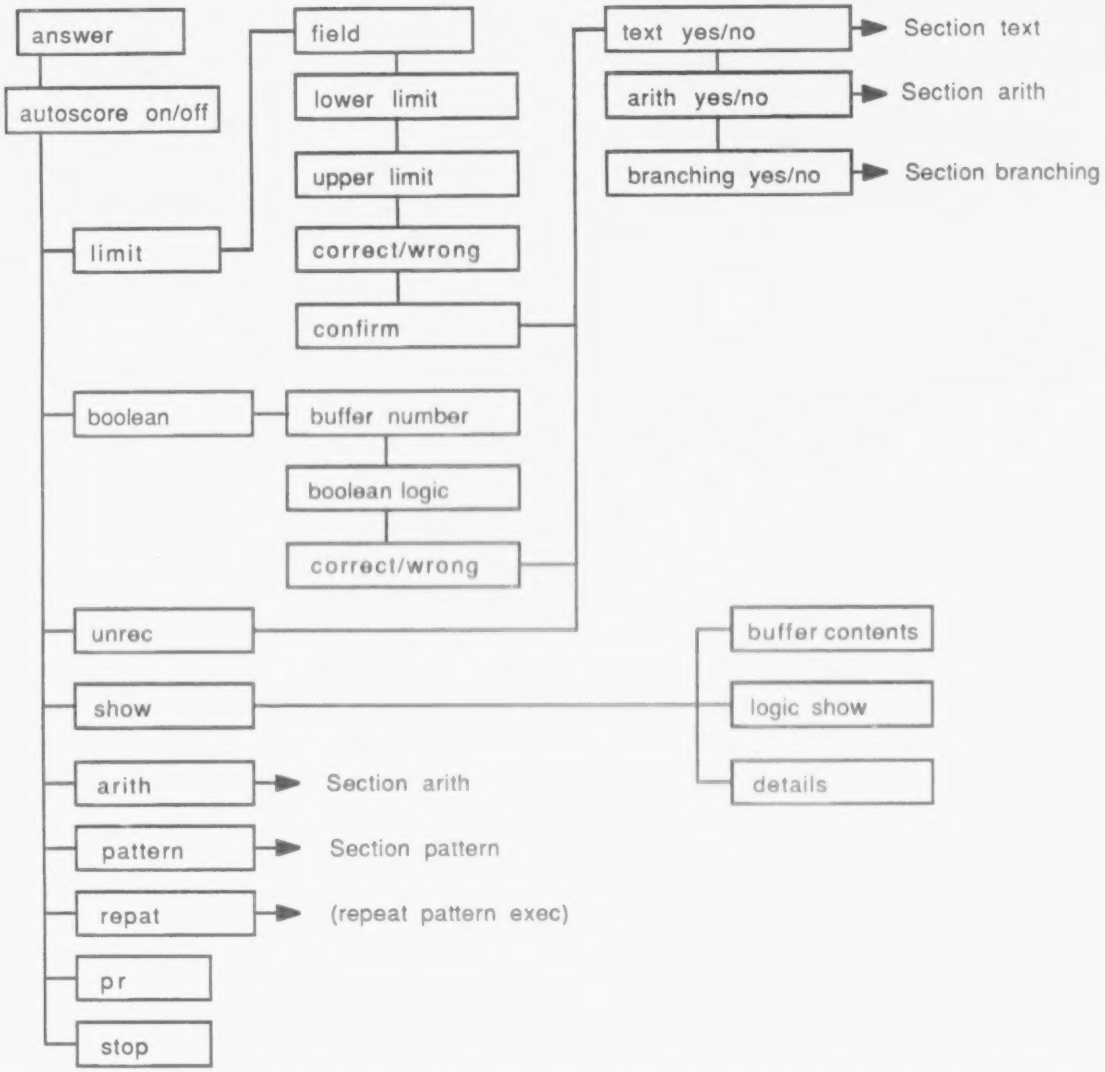
Provides some guidance in use of the options

stop

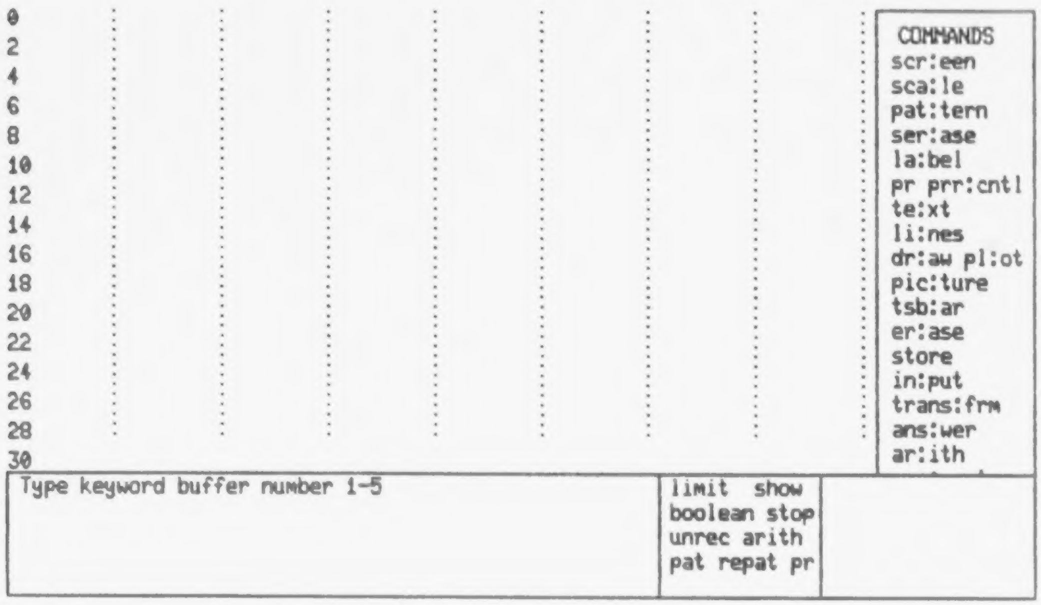
To return to the main menu.

Section 20
ANSWER

1.0 Introduction


answer
autoscore on/off
limit
field
lower limit
upper limit
correct/wrong
confirm
text yes/no
Section text
arith yes/no
Section arith
branching yes/no
Section branching
boolean
buffer number
boolean logic
correct/wrong
unrec
show
buffer contents
logic show
details
arith
Section arith
pattern
Section pattern
repat
(repeat pattern exec)
pr
stop

The author is first given the opportunity to set automatic score keeping on or off. Then the author is asked to specify whether the answer analysis is for a keyboard input or for a pointing input. The nature of the processes are different for each type of input.


0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
COMMANDS
scr:een
sca:le
pat:tern
ser:ase
la:bel
pr prr:cntl
te:xt
li:nes
dr:aw pl:ot
pic:ture
tsb:ar
er:ase
store
in:put
trans:frm
ans:wer
ar:ith
limit show
boolean stop
unrec arith
pat repat pr

```
Type keyword buffer number 1-5
```

When the input of the answer is via the keyboard, a boolean and numerical limit answer checking is provided. A Boolean analysis is based upon determining whether the student's answer contains selected words pre-stored in a buffer. (This procedure assumes that a set of keywords have been pre-stored by the author in a buffer. See STORE command in Section 17). The second method provided is for checking the numerical range of an answer, as is needed when one or more numbers is entered by the student. For example, a correct answer may be defined to be between 1.23 and 1.26.

In the case of a pointing response, an author is guided through the definition of rectangular target areas on the screen which are to be identified if pointed to by the student. The author defines these rectangular areas by using the cross hairs to mark the upper left and lower right extremities of each target area. The author must define whether the target being defined is to be a "correct", or "wrong", answer.

For both types of response analyses, the author may as an option call for a reply to the student, carry out arithmetic procedures, and define a branch. Normally, the author need not define a branch as an automatic branch will be made to the next instructional step.

During the process of defining the answer analysis steps the system will store sufficient information to allow a graphical representation of the steps taken. This is done for each type of answer analysis available to the author.

2.0 General Procedure for Answer from Keyboard

Generally, the process of answer analysis is as follows for a keyboard response:

1) Preliminary to accepting an input from the student any keywords to be used in Boolean analysis are stored. In the case of analyzing the student's numerical answer storing the numerical range of an answer is not required.

2) An input is accepted from the student

3) The transform option is selected and "boolean" selected if a string is to be analyzed

4) The "answer" option is selected. Automatic score keeping may then be selected. If a keyboard response is to be analyzed then the "limit" option, the Boolean option, or the "unrec" option is selected. (Normally, the "unrec" is the last option selected, i.e., after other answer possibilities are checked. There may be many unrecs in a series at the end of the answer analysis sequence.) The author specifies the boolean relationship to be tested, or the limits of a value to be tested.

4) With each answer option, "limit", or "Boolean" the author must provide a match-id identifier to identify into which answer category the student's answer falls. This identifier is normally "c" for correct answers, and "w" for wrong answers. A digit starting at 0 and ending at 9, and if more than 10 identifiers are required, the letters A to Z are automatically appended to c, and w, so that within categories distinctions are maintained, eg., c0, c1, etc.

5) The author then enters "text" mode and displays a message to the student. This is optional. If a message is given to the student, the author is queried for a note regarding this message.

6) The author may then take the "arith" option to carry out any arithmetic operations.

7) The author may then define a branch. The default is the implicit branching which is: branch to the next pr if a match-id started with the letter "c" i.e., correct (the pr is automatically generated upon leaving the answer mode); to repeat the input if the match-id was a "w" i.e., wrong, or "unrec". If a series of "unrec" have been sequenced, each will be executed in their rank order position for unrecognized answers, with the last "unrec" sequence repeated indefinitely. The author then may enter a comment to help identify the meaning of the branch.

3.0 Automatic Score Keeping Option

A menu headed with Auto Score provides the options "off", "on", and "go". If "on" is selected the automatic score keeping variables r*ight, w*rong, and u*nreg are created automatically if they have not already been created. These variables will be incremented by 1 each time a correct answer, wrong answer, or unrecognized answer, as defined in the answer analysis process, is encountered. If "on" is selected the automatic score keeping is enabled; when "off" is selected it is disabled. The author may use the score keeping variables for arithmetic purposes, but is not allowed to create these variables directly.

4.0 LIMIT Option

If the author wishes to check the numerical range of a student's answer, the "limit" option is selected. The author is then queried as to which buffer (0-5) the student's answer resides. Normally, the student's answer is in buffer 0, unless it had been ı queried for the following

Jote that any character which
al point will act as a delimiter
ə author to select the number
t, as would be required in a

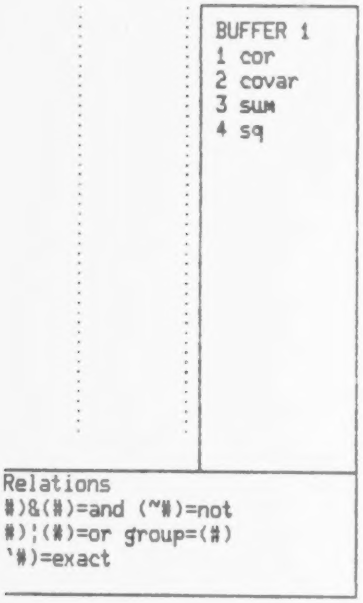
ult is minus infinity).

ult is plus infinity).

ɔng

əntered, and to change them
are acceptable. Parameters
ɣ an equal sign and the new

ıessage to the student, do



```
BUFFER 1
1 cor
2 covar
3 sum
4 sq

Relations
#)&(#)=and (~#)=not
#)¦(#)=or group=(#)
'#)=exact
```

ɔent's response is as follows:

Department of Englis
Graduate Guide

University of Alberta
Edmonton, Canada

1) The author must have earlier stored the keywords to be used in analyzing the student's answer using the "store" option of the main menu. Keywords should include elements which will be used to identify both the correct and wrong answers.

2) From the main menu, the options "input" and "transfrm" must have been called; in "transfrm" the "boolean" option would have been taken.

3) The "answer" option, in the main menu, followed by "boolean" is then selected. The author then defines a Boolean statement of an answer using the numbers associated with each keyword, and a set of defined relational operators. (A display of the buffer holding the keywords, with each keyword numbered, is made available to the author at this point). The steps are as follows:

    a) the author is asked which buffer (1-5) contains the keywords,

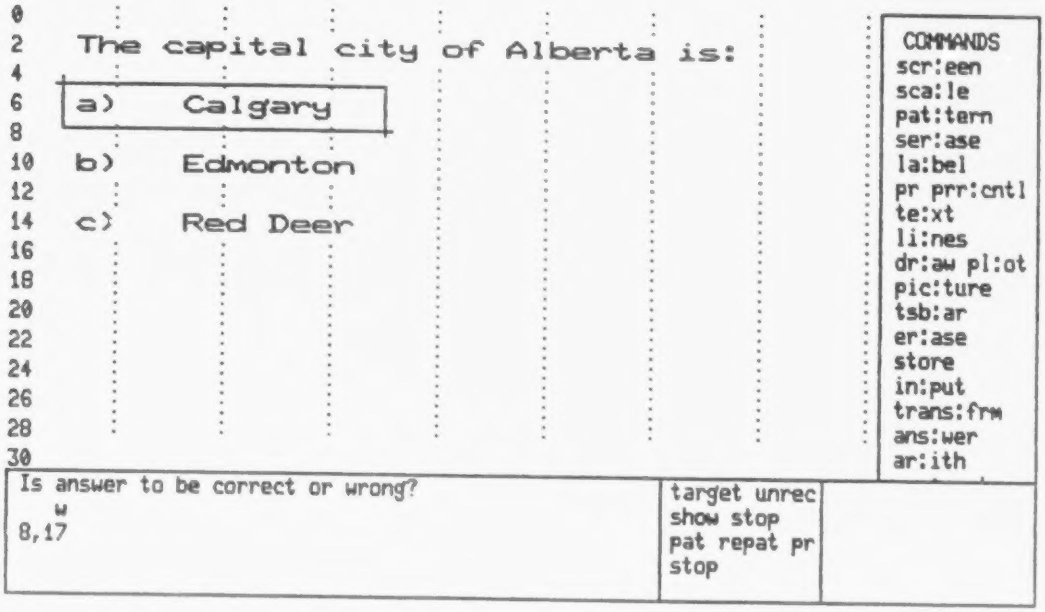
    b) the relations,(1)&(2) for "1 AND 2", (1)|(2) for "1 OR 2", (~x) for "NOT x" and (`x) for "EXACTLY x"; are displayed; (and) are used for grouping, and the author is asked to define a standard against which the answer will be checked. For example, to a question "Name the capital of Alberta?", and the keywords (1) edmonton, (2) alberta , and (3) calgary, the Boolean standard might be (1 & 2)&(~3), which means (edmonton AND alberta) AND (NOT calgary). The general form of the relationships are displayed in a menu.

4) The author is then asked to specify whether the answer is to be considered as correct or wrong, and taken through the options of providing a message to the student, doing arithmetic, and defining a branch.

6.0 General Procedure: Pointing Response

In the case of analyzing a pointing response, the author is asked to use the cross hairs to define the upper-left and lower-right of the target area of interest. The target area so defined is outlined on the screen for the author. The author is then asked to identify whether this area is to be considered "correct" or "wrong". From this point on the same information is requested from the author as for a keyboard response, and a flow chart of the logic created is also made available. The author may also define an unrecognized pointing response. In this case the area does not have to defined, i.e., it is a residual category and handled in the same manner as for a keyboard response. It is suggested that at least one row (half line) be left between target areas. A translation has to be made between the pixel coordinates of the Gigi screen, and the row specification of the student's screen.

An example of a screen displayed during the definition of a pointing response follows.



```
0
2   The capital city of Alberta is:          COMMANDS
4                                            scr:een
6   a)    Calgary                            sca:le
8                                            pat:tern
10  b)    Edmonton                           ser:ase
12                                           la:bel
14  c)    Red Deer                           pr prr:cntl
16                                           te:xt
18                                           li:nes
20                                           dr:aw pl:ot
22                                           pic:ture
24                                           tsb:ar
26                                           er:ase
28                                           store
30                                           in:put
                                             trans:frm
                                             ans:wer
                                             ar:ith
Is answer to be correct or wrong?    target unrec
   w                                 show stop
8,17                                 pat repat pr
                                     stop
```

7.0 UNREC Option

The "unrec" option provides a residual category in the sense that it is executed when a correct or wrong answer cannot be identified. There are no parameters associated with the process. The author is given the option of defining a message to the student, do arithmetic, and to define a branch identical to the definition of a correct and wrong response identification.

8.0 SHOW Option

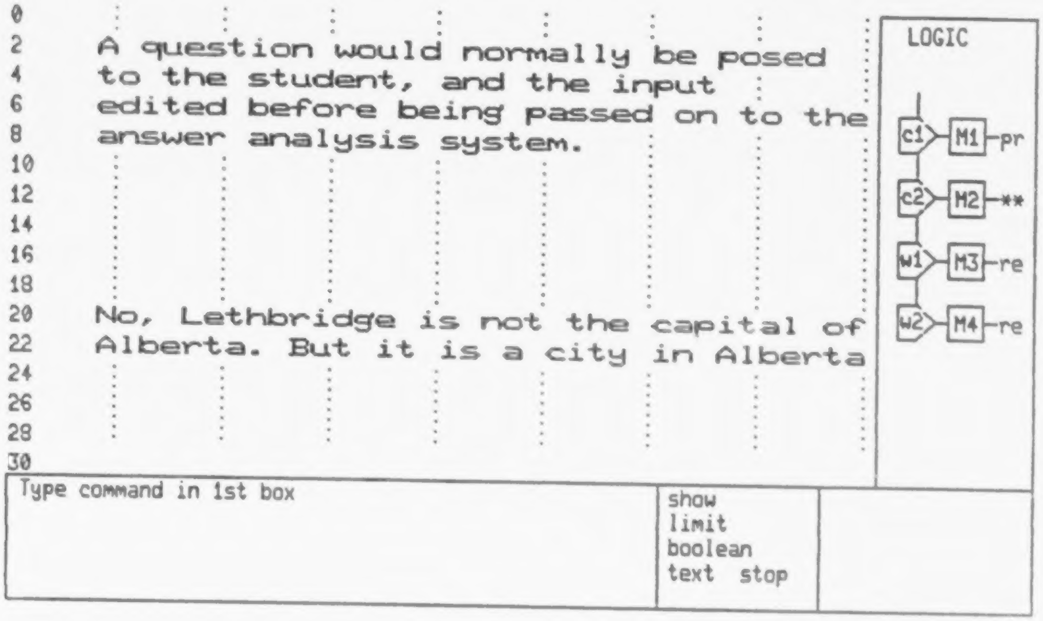
Since the process of checking an answer can be complicated, the author is provided with some support to review what has been done. Selecting the option "show" will provide the author with the commands "buf", "logic", "details" and "stop". These have the following actions:

1) **buf:** - allows a buffer to be displayed. The author can review the keywords which were stored under the store option.

2) **logic:** - displays a logic chart of the answer analysis process as created by the author (see next illustration). In the logic chart the first box starting from the top contains the first procedure created and its identifying match-id; the next box will contain an "M" if the message option was taken by the author; following this box will be an indication of what branch is being taken e.g., pr, re, or **. pr means branch ahead to the next pr defined, re means return for a repeat of the input, and ** means a special label to branch to was entered.

3) **details:** - allows the author to ask for more details about the actions represented by the logic chart noted above under "logic". Typing the match-id associated with any one of the procedures diagrammed (the match-id is located in the box of the flow chart, e.g., c0) will provide the author with the comment entered about the message, and the branch which will be taken under execution. The display of logic details is terminated by typing the word "stop".

The amount of screen space is very limited, and therefore the amount of detail regarding the answer analysis procedure is restricted. In addition, no provision has been made to remind the author to eventually define branch-to points (labels) defined in the answer analysis logic.


A question would normally be posed to the student, and the input edited before being passed on to the answer analysis system.

No, Lethbridge is not the capital of Alberta. But it is a city in Alberta

Type command in 1st box

9.0 Pat, Repat Options

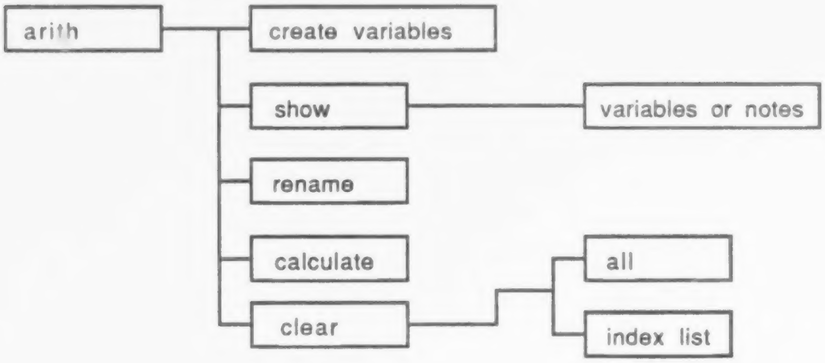
Since the answer analysis procedure tends to be highly algorithmic, access to establish a pattern of the process can be made by taking the option "pat". (See Section 8). Once the first pattern has been created, it can be re-executed by using the option "repat". For example, through the option "pat" a check on the a correct answer might be made with "rem" enabled. Further correct answers could be checked by the option "repat". The pattern defined for checking a correct answer could be edited (changing c for w) and then used to check for wrong answers.

10.0 pr

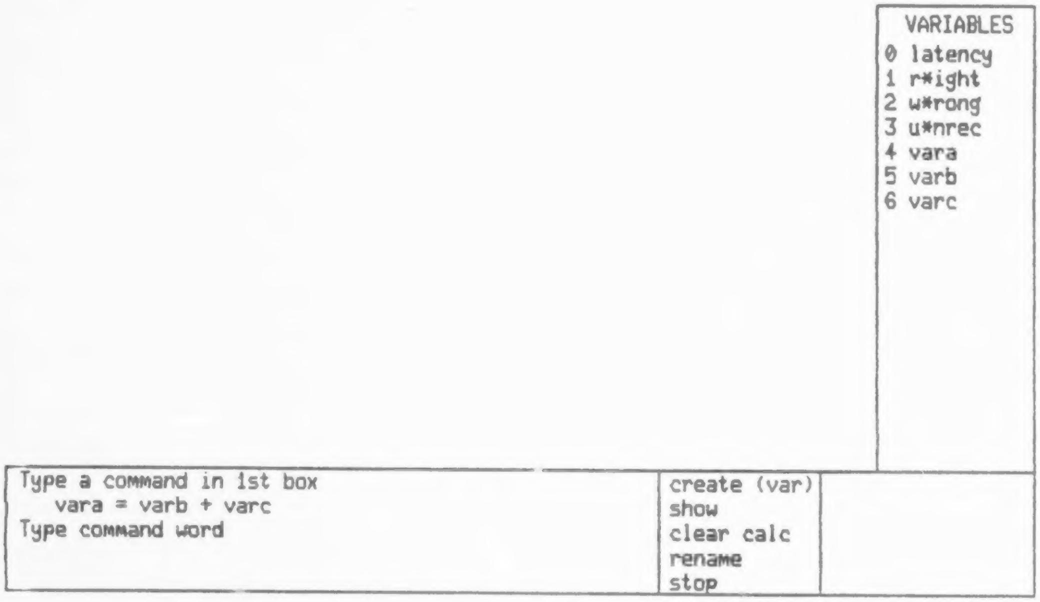
It is sometimes required that the author have control over the positioning of the pr for checking student answers. Selecting the option "pr" causes a pr to be created for purposes of implicit branching.

Section 21
arith

1.0 Introduction


arith
create variables
show
variables or notes
rename
calculate
clear
all
index list

The "arith" option provides the author access to carry out simple arithmetic calculations. Such calculations are useful for keeping track of such events as the number of questions answered correctly and incorrectly. The authoring system interfaces the author to 32 integer counters which are named by the author. These variable names can be renamed. To assist the author to keep track of the purpose of each variable, the author may enter a 40 character comment about each one.



```
VARIABLES
0 latency
1 r*ight
2 w*rong
3 u*nrec
4 vara
5 varb
6 varc

Type a command in 1st box                create (var)
   vara = varb + varc                    show
Type command word                        clear calc
                                         rename
                                         stop
```

The commands provided with the "arith" option are as follows:

create	to create a new variable, or to create the automatic score keeping variables; duplicate names are not permitted.
show	to display to the author the names which have been created and to display notes about them.
rename	allows the author to rename a variable; if the author exceeds 32 variables and needs additional ones this can only be done by renaming ones already used. (Actually, 5 additional variables are defined as external variables through the student registration record.)
calc	allows the author to define simple arithmetic expressions of the form a = b operator c.
clear	allows the author to set variables to the value of zero
stop	to return to the calling menu.

The sequence of events invoked by each of the commands noted above are as follows:

create: The author may have the automatic score keeping variables r*ight, w*rong, and u*nrec created, or a new variable may be created provided that 32 variables have not already been created. A variable name is accepted if it starts with the letters a to z, and followed by 0 to 7 of a to z or 0 to 9. For example, the variable names right, wrong, a12, z99 would be acceptable. The existing variable names list is checked for duplication, and if no duplicate exists the name is established (and transparently assigned a counter which the author need not be conerned about). The author is than asked to provide a note about the new variable (1 to 40 characters may be provided).

show: The author is asked whether "variables or notes" are to be displayed. The "variables" option displays for the author the names of the variables already defined. The "notes" options allows the note associated with each variable to be displayed. This means that a display of the variable names should be done first if the author is not sure what variables names have been defined.

rename: The rename option requests the author to define the name of the variable to be replaced, then the new name by which it should be replaced, and then a note to be associated with the new name.

calc: This option allows the author to carry out simple arithmetic expressions of the form a = b operator c, where operator may be + - x and /. The form a = number is also acceptable, i.e., to initialize a variable to a numerical value, e.g., right = 0, constant = 5.

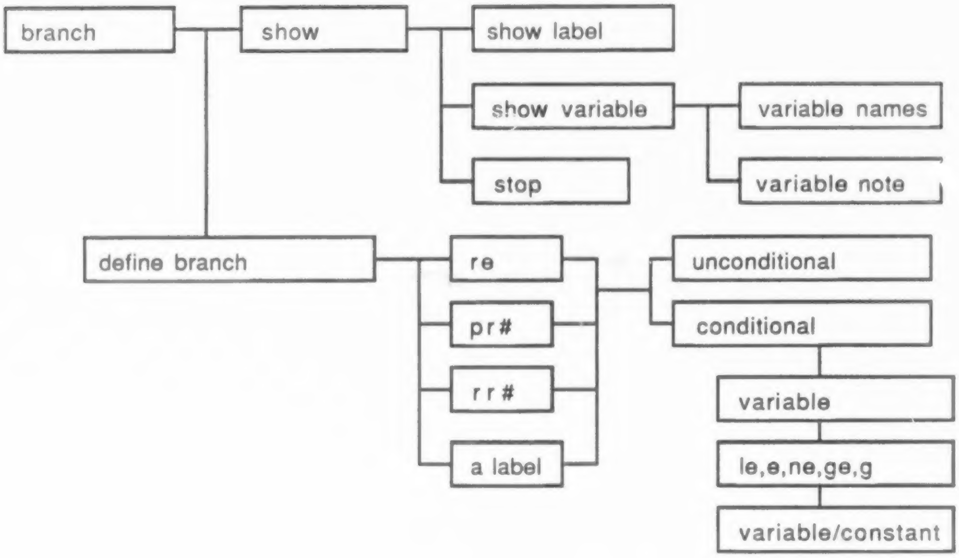
clear: Using this option the author can cause all variables, a list, or range of variables to be set to zero. An indexed list of variables is displayed from which the author can specify those to be set to zero, e.g., 1,3,4; 2,3,10 to 12.

Section 22
comment

This option allows the insertion into the stream of code generated the following statement: "nop #COM comment" where the "no" means no opcode. This is useful if the author wants to mark a location in the code generated for some purpose, eg., to make correction using one of the system editors. Using the editor, a search would be made of #COM to locate the desired comments.

(Note: the editor called by the option in the main menu is not a system editor and does not have a "search" capability). However the comment may be used to mark a point in the code where changes may be required.

Section 23
branch


branch
show
show label
show variable
variable names
stop
variable note
define branch
re
pr#
rr#
a label
unconditional
conditional
variable
le,e,ne,ge,g
variable/constant

Provision is made to allow the definition of unconditional and conditional branching. Conditional branching can be made dependent on the usual relational operators relating two variables or a variable and a value.

Branching is allowed to "re" (return to last input), to "pr(n)" to the nth pr, to "rr(n)" (return register), or to a label. Since authors might wish to define the label stop, exit from the grammar is provided by "*stop". The previous pr is 'pr0', the next pr is 'pr1'. Branching to a "pr" should be used cautiously since they are relative addresses, and their encounter during execution may be hard to determine in a complex program. Explicit branches to labels are more meaningful, less confusing, and less prone to causing errors.

The sequence of commands available are as follows:

show: allows for the display of label names or variable names

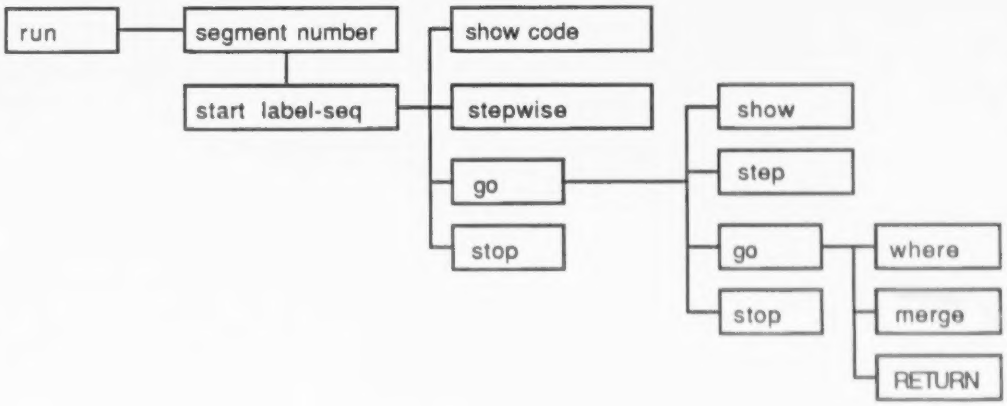
define branch: allows for a branch to be made to the following:

re	repeat input
pr#	branch to a pr code
rr#	branch to label in return register #
a label	

conditional or unconditional: allows for branch to be made based on the relationship between a variable value and another variable or a constant, or to branch without any conditions. An example of a conditional branch is branch to label xyz if variable right is greater than 5. The relationships allowed are as follows: l (less than), le (less than or equal), e (equal), g (greater than), ge (greater than or equal).

Section 24
RUN Option

1.0 Introduction



The "run" option provides the author with the following features:

1) to execute code which has been created so that it can be examined as a student will see it

2) to show the code which was generated by the authoring processes

3) to mark the point in the code of one segment at which the code of the currently authored segment will be inserted

2.0 Use of the RUN Option

This option was created for immediately testing the adequacy of the course being generated, and in the case when very long sections of a course are being created, to create smaller, independent units for insertion after testing, into the longer section. However, the option may also be used to insert new material into an existing section directly, since upon return from the use of this option, editing may continue.

The author must carefully consider the strategies of course creation when lengthy courses are being created. The run option may be used in the manner described below.

3.0 Appending to an existing Segment

The author may append new instruction to the end of an existing instructional sequence. The author may take the option "run" to execute and view the instruction which was created at the last session and then move into authoring. This provides the author with maintaining continuity for the topic in the authoring process. The newly authored material is appended to the previous course contents.

4.0 Merging new Content into an Existing Segment

New content may be inserted into an existing segment at a specific point through the use of the "run" option. The following example illustrates the use of the merge feature. We assume for purposes of illustration that segment number 123 exists and that new instruction is to be created and inserted at a particular point in segment 123. The author could author, say segment 124, creating the new instruction. The author would then run segment 123 to the point where the new instruction from segment 124 is required, indicate that at this point a merge from the currently authored segment (124) is to be made. Upon return to the authoring mode segment 123 will contain the new instruction. The author may continue to create more new instruction in segment 124 and inserting it into any other segment. In order to prevent the creation of duplicate labels, the author should number segments in at least steps of 2, and use one of the segments solely for purposes of inserting new instruction. For example if segments 1 and 2 were created segment 2 might always be used to create inserts into segment 1. Since new labels are likely to be created in segment 2, any authoring on segment 1 should rebuild the label table (use the build option instead of read option when the author is queried about re-specifying the registration parameters immediately after signing on).

After typing the command run (in the main menu), the author is prompted for the number of the segment to run, and then the label at which execution is to begin. The author may run the current segment or a different segment.

The author is then queried as to which run characteristics are desired. These options are:

show code:	will display code being executed
stepwise:	to execute only one instruction at a time
go:	to start execution
stop:	to not proceed with the run option

During execution when the stepwise mode is used, after an instruction is executed the following options are available:

1) The RETURN key will cause the next executable instruction to be executed

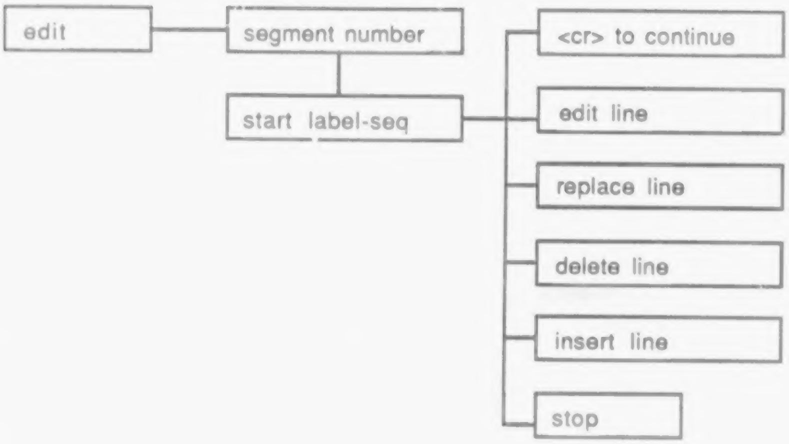
2) **where** will indicate the label-line number of the current instruction executed

3) **merge** will mark the point at which the segment which was being authored prior to invoking the run command will be merged

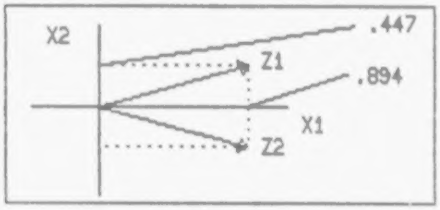
4) **stop** to stop further execution

The normal exit from course execution used by the student, i.e., typing the @ symbol will display the interpreters usual set of commands at this point. If the end of the course has been reached during execution, this will be noted by the interpreter.

Section 25
edit


edit
segment number
start label-seq
<cr> to continue
edit line
replace line
delete line
insert line
stop

The edit option allows the author to edit certain statements generated by the authoring system. The editor is invoked by the command "ed" or "edit".


X2
.447
Z1
.894
X1
Z2

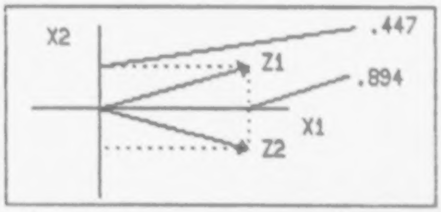
The projection of Z_1 on X_1 is .894. The correlation of Z_1 with X_1 is .894. The same is true for Z_2.

The author is asked to provide the segment number to be edited, and then the label at which editing is to begin. The segment to be edited may be the current segment, or a different one. Each line within the boundaries defined (from the label at which editing is to begin plus 150 more lines) are placed into a list structure for editing. Each line as it is encountered is displayed at the bottom of the screen, and in addition, the following instructions are executed by the interpreter: db, dc, de, dt, dl, fp, drw, sv0, sv, ep, and rgs. These are the instructions which cause visible screen results. The other instructions which are displayed but do not cause visible screen results, may be edited but are not executed by the interpreter. After150 lines have been displayed, the procedure will repeat the sequence so that the author can be make further checks and changes.

Once a line of code has been displayed, the author may use one of the following commands as the next step:

1) depressing **RETURN** key will cause the next instruction to be brought in, i.e., to continue

2) **edit:** will cause the currently displayed instruction to be re-displayed in cyan, and the cursor positioned at the beginning of the line. The line is now ready for editing. The arrow keys are now in effect to move the cursor to any position in the line. Characters may be deleted by using the delete key; characters may be inserted simply by typing the desired character. Insertion occurs in front of the cursor. To exit the edit mode, type CTRL z (hold down the key marked CTRL and type the letter z). The edited line will be executed so as the effect of any change may be observed.



The projection of Z_1 on X_1 is .894. The correlation of Z_1 with X_1 is .894. The same is true for Z_2.

```
db  16,2/8,43,24/\cyThe projection of Z\ii1\iron X\ii1\ir is .894. The\3ncorrelation
 of Z\ii1\ir with X\ii1\ir is .894. The\3nsame is true for Z\ii2\ir.<
```

3) **replace:** replace will cause a line of code to be removed allowing the author to enter a new line. Type CTRL z after the new line has been typed to return to viewing the next statement.

4) **delete:** delete will cause the currently displayed line to be deleted.

5) **insert:** insert will allow the author to insert a new line of code immediately after the one being displayed. Use CTRL z after the new line has been typed to return to viewing the next statement.

6) **stop:** will cause the edit mode to terminate, with a return to the main menu of the authoring system.

Section 26
EXIT Option

The "exit" option signals that the end of this authoring session is to be made. At this point information is written to at least two files which then form the basis for restarting the next authoring session. One file contains information pertinent to the whole course (such as drawing names, buffer names, etc.), and another file contains information pertinent to the segment (e.g., labels, variable names).

Section 27

Program Code Generated

Most authors will not want to examine any program code. For those authors wishing to use existing editors under the VAX operating system, an understanding of the code generated by the authoring system will be helpful. The following is a brief description of the code:

1) ** defines a label, e.g., ** xyz defines label xyz

2) **pr** defines a point to which automatic branching will be made

3) **prr** same as pr, but it also forms a student restart point at next signon

4) **db** defines the text to be displayed, the first 2 parameters are starting row (sr) and starting column (sc), the next two are number of rows (nr), and number of columns (nc); the 3rd is the number of rows to add to get the next available row.

5) **de** defines an erase; 1st parameter is starting row, second is number of rows to erase (selected area erases, ie., not over the full screen width, is done by displaying a blank using the db with appropriate parameters)

6) **fp** displays fp-type drawings

7) **drw** with identifier followed by row/col to indicate where drawing is to be placed on the screen.

8) **pa** is pause followed by time of pause

9) **pae** pause until a key is touched

10) **ep** position cursor (same screen position parameters as in db) and enable keyboard for input; additional parameters are for number of characters to input before continuing with the execution of more code; last parameter is the response identifier used to identify the location of input in the course

11) **epp** pointing response; 1st parameter is time; last is response identifier

12) **fn2** defines a special function to carry out special processes used in answer analysis; the specific process is defined by the next 2 to 3 characters; the parameters vary from function to function depending upon the process

13) **un** identifies an unrecognized answer check; a sequence of un codes are executed on the 1st, 2nd, 3rd unrecognized input of an answer; the last un is always executed when all earlier un codes have been encountered.

14) **ad** is an addition code; addition is made of a constant to a counter, or a counter to a counter

15) **sb** is for subtract

16) **mp** is for multiply

17) **dv** is for divide

18) **ld** for loading a constant into a counter, or loading the contents of one counter into another counter; also used to load text to a buffer (ld text/buffer number).

19) **br** is a branch code; branching may take the following form: branch to a label, return register, to get input again (re) or to a pr; branch may be conditional; branch to a label, return register, to input, or to a pr if a counter is le,e,ne,ge,g than the contents of a counter or some constant.

20) **tr** is for branching to a new segment; transfer segment/label causes a transfer to a new segment; it may also take the form of tr next/label where the next segment is found in the student's registration file.

An example of code generated by the authoring system is given below:

```
**  stat4:coursename
**   start:start label
** 107000:first numeric label
de  0/32
db  4,3/4,42,8/\cyThis is an example of placing text on\2nthe screen.
db   10,3/2,42,12/\cyEquation:   Mean = \1fa\0fX\1ii\1i\2b__\2i\1bN
db  30,13/2,2,0/\cgTouch Space Bar\cy
pae
rgs   w(i3)p[210,107]v[][+220][,+86][-220][,-86]
rgs    w(r)w(a0)w(i6)w(p1m2)
db  18,3/4,42,22/\cyExamine the equation for the mean in\2nthe box shown above.
db  30,13/2,2,0/\cgTouch Space Bar\cy
pae
de  0/32
db  2,3/6,42,8/\cyWhich of the following is the equation\2nfor the mean? (a,b, or
c)\2n db  8,5/2,40,10/\cya) \1fa\0fX/N   b) \1fa\0fX+N   c) \1fa\0f-N ld  a b c/b1
epi   12,3/2,12/7,3//3/107000-016
nx
br  re
fn2 cod/1,1
fn2  sq/c0/(1)
de  16/14
db  16,3/2,42,18/\cyYes, a is the equation for the mean.
ad  1/c2
fn2  sq/w0/(2)
de  16/14
db  16,3/2,42,18/\cyNo, b is not the mean.
un
de  16/14
db  16,3/2,42,18/\cyWhat? Just type a, b, or c.
pr
db  30,13/2,2,0/\cgTouch Space Bar\cy
pae
en
```

Section 28
Execution of the Course Authored

1.0 Introduction

The program code file which is created must have the labels "coursename", "start", and a label made up of the segment number and 000, as the first three labels. This would normally be done automatically by the system at the beginning of the first authoring session. The last opcode automatically written by the authoring system will be an "en" (end of course) and is used by the interpreter to identify the end of the course code. The exact method of executing the course will depend upon the privilege level of the author. Authors with privilege level 0, are provided with a menu which allows selection of an execution mode. Authors with privilege to work within the context of the operating system may use the procedures described below.

During the author's first session, execution can be made of the course by calling up the execution option in the author menu. However, the execution of code produced at subsequent sessions depend upon how the author wishes to proceed. The following strategies can be used:

1) The author may define the segment of the initial authoring session to be a "test segment" which will be renamed to its correct segment using VMS commands. For example, for the course "math" the author may initially author segment 9. This will produce a file math009.cw2n. This file may be edited using the EDT editor. Once the code executes properly it may be copied to math001.cw2n. Subsequent sessions may also be for segment 9, and when tested and found to be working adequately its first two labels (coursename and start) should be removed. Then this file may be appended to math001.cw2n. However, the last opcode in math001.cw2n before the appending would be an "en", and this would have to be removed using the EDT editor. This strategy avoids the accidental loss of tested code through the use of the VMS purge command. However, labels automatically assigned will have the first 3 digits equal to the segment number being authored. An author may wish to change these to the segment number to which the file is eventually concatenated.

2) The author may wish to author using the same segment number each time. Care must then be taken as to which version of a file is appended to the initial version, and a purge must not be done until the appending has been completed. The author must keep in mind the name of the first label defined during the current session, as this label will be the point at which execution is likely to be started (otherwise, execution may have to be started at the label coursename or start which may require the author spending considerable time approaching the new content from the beginning of the segment).

Section 29
Files Used by The Authoring System

1.0 Introduction

The authoring system makes extensive use of files within the VAX system. The file system is very much like the file system under the University's MTS system, but there are some differences.

Normally, the author need not be aware of the files being created, however, some information is provided here to give the author a better understanding of the authoring system.

2.0 File Names

The names of files being used for the authoring process are defined within the grammars themselves. Files within the system have the following general format:

1) **filename.extension;version**

where
filename is the name of the file either established by the application grammars, or by a user.

2) **extension** is a further designation, or sub-name of the file.

3) **version** is the version number of the file. Normally each time the same named file (same in the sense of filename and extension) is modified the new and old versions are kept, with the newer version number being one greater than the previous version.

3.0 Files Used by The Authoring System

The following files are used by the authoring system:

1) **xas.auf** This file will contain parameters to help tailor the authoring system to the individual author. It contains a privilege level, whether new course code should be appended to existing course code, whether a list of labels should be established from the existing course code or from authoring information saved from the last authoring session, and whether all commands used by the author in the processing of authoring should be saved.

2) **crseg.ycmd** ; This file will contain the commands used by the author during the authoring process. It is being collected for research purposes. The designation crseg stands for "course segment" or "course chapter".

3) **crseg.yap** ; This file contains information for the continuation of an authoring session. The specific information which it contains is as follows:

a) a list of each label in the course as well as up to 40 characters of comment

b) a list of those labels to which a branch has been made but the location of the branch is yet to be defined and up to 40 characters of comment

c) variable names and up to 40 characters of comment; the first variable is automatically entered as "latency" which is reserved for holding the student response latency

d) course code data describing the contents of certain data structures in the course, e.g., keyword buffers which have been loaded for answer analysis purposes.

4) **crseg.cw2n** ; This file holds the program code generated by the system. Depending upon the parameters set in the author's xas.auf file, newly created code may be appended to course code which already exists, or placed in a new version of the crseg.cw2n file. If the code for a current session is not automatically appended to existing code, the author will have to concantenate the most current file to the previous file using VMS commands.

This file may be edited by the EDT editor under the VAX VMS operating system. The file crseg.cw2n is also the file which is submitted to the interpreter for execution so that the author may test the actions of the course as a student would observe them.

5) **drwxxx.reg** ; A file of this form is created when a drawing is created. drw indicates that the drawing is relocatable, and xxx is a unique identifier selected by the author (usually 3 alphabetic characters). The identifier drwxxx and up to 40 characters of comment are saved for use by the author during the authoring process.

6) **fpnnn.reg** ; A file of this form is created when a drawing of the "slide" type (old IBM 1500 film projector frame) is constructed. It is displayed in a fixed position on the right-hand side of the screen when the screen is in a split-frame mode ("split" option in the command "screen" of the first menu).

7) **crs.ycon** ; This file contains the following information:

a) a list of all ids of drawings created and up to 40 characters of comment

b) a list of all the pattern names (instructional templates) created and up to 40 characters of comment

c) a list of all the keyword buffers created and up to 40 characters of comment, as well as the contents of the buffers.

8) **xxxxx.ypat** ; This file contains the author commands for the specific template named "xxxxx" created and saved by the author. The author specifies a unique name (xxxxx) under which the template is saved during the authoring process.

4.0 Authoring Program Files

The directory [steve.ausys] holds the object grammars which run the system. This directory also holds the file cai.com which contains all the file assignments which are required. The latter file also contains the assignments needed for the support systems, since the entry to authoring is made through the existing CAI support system.

CAUTION: The entry to authoring presents the author with a menu which allows access to Proctor Functions. Upon leaving the Proctor Function mode, 2 records are written out which are normally appended to the registration file (user.urf). However, this is not done automatically as for a normal proctor at signoff. Thus, an attempt to have the user.urf file read will cause an error as the 2 line record which is available does not have the full registration information. (Temporary problem).

Section 30
Using the External EDT Editor

Files produced by the authoring system may be edited by the EDT editor. EDT is a powerful screen editor with many commands. If the author's privilege level allows access to files through the operating system, the author may edit any file directly. When the privilege level is 0, editing of only the course code file is allowed.

In the EDT editor commonly used editing keys are as follows:

1) **ARROW** keys - move the cursor within screen boundaries in direction of the arrow.

2) **DELETE** key - deletes the character to the left of the cursor.

3) **BACKSPACE** key - moves cursor to beginning of line (far left).

4) **6 on keypad** - deletes character the cursor is on moving the characters from the right with each deletion

5) **2 on keypad** - moves cursor to end of line (far right).

6) **PF3 on keypad** - deletes the remainder of the line from the cursor to the end of the line including the carriage return

7) **PF1 and PF3 on keypad** - causes the last line deleted to be undone.

8) **- on keypad** - pages ahead one screen frame in the file.

9) **+ on keypad** - pages back one screen frame in the file.

Some special key combinations are also available:

1) **PF1 and 8 on keypad** -enters "search for" mode; after typing the string to search for use the ENTER key (NOT the RETURN KEY); use 8 on the keypad to search again;

2) **4 followed by 8 on the keypad** will cause forward searches,

3) **5 followed by 8 on the keypad** will cause backward searches.

4) **PF1 and 4 on keypad** displays the last part of the file

5) **PF1 and 5** displays the beginning part of the file

6) **CTRL and z** -places the cursor at the bottom left of the screen ready for the following special commands:

a) **set notruncate** - allows the screen to display lines normally truncated on the screen because of their length. Lines are not truncated in the file being edited.

b) **exit** - causes exit from the screen editor WITH A NEW VERSION of the file being created containing the changes which where made.

c) **quit** - causes exit from the screen editor WITH·NO NEW VERSION being created. Changes made to the contents of the file WILL NOT BE SAVED.

d) **t** - will display the line number at which the cursor is currently positioned.

e) **c** - will cause re-entry into the normal edit mode; the cursor will be positioned at the point at which the CTRL z was done.

6) ALTERNATIVELY to the use of **CTRL and z,** one can hold down the PF1 key and depress the number **7 key on the keypad**, to allow entry of one command described under point (5)

During the editing process, the **PF2 key** will provide access to an on line help system for using the editor.

Section 31
Startup of the Authoring System

1.0 Initialization of Files

To start authoring the author must have a signon ID and password on the VAX system. Jim Hunka, or Vladimir Bado carries out this registration. The registration must include the name of the course to be authored, eg., stat4, read1, math2, etc., as this name is used by the system. The author should also indicate the numbers of the chapters being created, and whether the course will have an internal table of contents.

The file xas.auf containing the following, must also be available:

1) privilege: 0 or 9
2) command: yes, no, or query
3) labels: build,read, or query
4) append: yes, no, or query
5) comment: any comment

Student authors have their privilege set to 0; "command" refers to the saving of all authoring commands; labels can either be "build" from the last course code file, or "read" from the crseg.yap file; yes for append will append the course code of the current session to the code of the previous session. The term "query" if encountered under privilege 0 will cause the command parameter to be set to "yes", the labels parameter to be set to "build", and the append parameter set to "yes". In the case of privilege level 9, the author is queried for the setting desired for the current authoring session. A check of the logic of the parameter settings is made, and when required the parameters are reset for the currrent session. For example, if a course does not yet exist, the parameter "yes" for append will be changed to "no". If the parameter "build" is used for labels, the existing course code will be examined and all labels identified and placed into a list for guidance of the author during an authoring session.

Note that the file userid.urf is to have global switch [3] set to 1 for a student author (privilege 0). This governs the display of certain entry menus.

2.0 Logical Assignment File

The ELF grammars used by the system reside in a directory of files outside the file system of the author. To access these grammars a file of "logical assignments" must be established to link the author's grammar requirements to the location of the grammars. This may be done by typing @cai. Typing @cai must be done prior to starting each authoring session if its invocation is not automatically done through a login process.

3.0 Down Loading Character Fonts

If the author is using a particular character font, this may have to be loaded to the terminal. For example, to load the font used in the statistics course, one would execute the following command:

```
$type [courses.stat2]allstat1.fnt
```

and wait a few seconds while the font is being loaded.

4.0 VAX Computer File Names

The authoring system makes use of files for the storage or course code, and that information which is required for continuation of authoring sessions. Depending upon the privilege level of the author, some understanding of the file system may be important.

The VAX operating system is oriented towards handling files. File names have the following form:

filename.extension;version

where,

1) filename is user defined or defined within the application grammar (program)
2) extension is user defined or defined within the application grammar
3) version is a number starting at 1 and incremented by 1 each time the file is modified or written to by the application program or editor.
4) the period and semicolon must be present

For example, each authoring session might produce a program code file math001.cw2n;1, math001.cw2n;2, etc., for the course called math. The 3 digits following the course name (i.e., 001 represent the segment number of the file). Student-authors have their file automatically created as userid000.cw2n with a version number, and the course code for a current session is appended to the code from the previous session.

5.0 Operations Related to Files

The following commonly used commands are part of the operating system of the VAX computer:

1) **dir**; produces a brief list of all files for the current directory
2) **differences filename1 filename2** ;lists to the screen the differences between filename1 and filename2
3) **append filename1 filename2** ; appends filename1 to the end of filename2
4) **purge** ; destroys all but the last version of all files in the current directory
5) **purge filename**; destroys all but the last version of the file called filename.

(NOTE:The * is "wild" and can be used in place of a filename, an extension, or a version number. For example, purge read1.* would purge all but the last version of the files read1 regardless of their extension.

6) **delete filename.extension;version** ; destroys the file named. The version cannot be omitted in this command.

More information about these and other commands can be obtained by typing "help" while the symbol $ is on the screen to the left of the cursor.

Authors who have access to the operating systems commands, must call the editor by typing the following command:

edt filename.extension, e.g., edt read1.cw2n

When the version number is not used, the last version will be edited. If a file which does not exist is edited, a message will be given that the file does not exist, but entry to editing mode is made.

research and information report

RIR-88-3

APPENDIX C
SYNTAX OF IMPLEMENTATION TWO

COMPUTER ASSISTED INSTRUCTION AUTHORING SYSTEM

SSHRC PROJECT NO. 410-84-1111-R2

S. Hunka

division of educational research
faculty of education
the university of alberta
edmonton, canada

RIR-88-3

APPENDIX C
SYNTAX OF IMPLEMENTATION TWO

COMPUTER ASSISTED INSTRUCTION AUTHORING SYSTEM

SSHRC PROJECT NO. 410-84-1111-R2

S. Hunka

Section C1	Identifiers, Types, and Expressions
Section C2	Control Language Syntax
Section C3	Content Language Syntax
Section C4	Display Language Syntax
Section C5	Answer Language Syntax
Section C6	Input Language Syntax
Section C7	Menu Language Syntax
Section C8	Expressions Syntax

Division of Educational Research Services
Faculty of Education
University of Alberta, Edmonton
February 1988

Syntax Developed by:
W. Chiu
T. Garraway
J. Higham
N. McGinnis
J. Nesbit

Table of Contents

Page
Section C1 1
Identifiers, Types, and Expressions 1
1.1 Introduction 1
1.2 Identifiers 1
1.2.1 Scope Of Identifiers 1
    1.2.2 Case Sensitivity Of Identifiers 2
1.3 Routines & Functions 3
1.4 Restart_points & Control Modes 3
1.5 Types, Variables, & Constants 4
1.5.1 Types 4
    1.5.1.1 System Types 5
1.5.2 Variables 6
    1.5.2.1 Dynamic Variables 6
1.5.3 Constants 6
    1.5.3.1 System Constants 7
1.6 Expressions 8
1.6.1 Numeric Expressions 8
    1.6.1.1 Numeric Conversion Rules 8
1.6.2 String Expressions 8
1.6.3 Operands 9

Section C2 11
Control Language 11
2.1 Introduction 11
2.2 Syntax 11
    2.2.1 Process Body 12
    2.2.2 Simple Statements 13
    2.2.3 Process Calls 14
    2.2.4 Assignment Statement 15
    2.2.5 Structure Statements 15
    2.2.6 Expression Evaluator 17
    2.2.7 String Expression Evaluator 17
    2.2.8 Numeric Expression Evaluator 18
    2.2.9 Qualified Data Element Evaluator 20

Section C3 22
Content Language 22
3.1 Introduction 22
3.2 Syntax 22
3.3 Process Body 23
3.4 Simple Statements 23
4.5 Process Calls 24

Section C4 26
Display Language 26
4.1 Introduction 26
4.2 Syntax 26
4.3 Process Body 27
4.4 Common Simple Statements 27
4.5 Display Simple Statements 28
    4.5.1 Textbox And Related Statements 29

4.5.2 Embedded Text Commands 33
4.5.3 Drawbox And Related Statements..........................34
4.5.4 Embedded Draw Commands..................................36

Section C5 37
Answer Language 37
5.1 Introduction...37
5.2 Syntax...37
5.3 Categories..38
5.4 Acon Action 39
5.5 System Routines, Functions, And Response Analysis.....39

Section C6 44
Input Language 44
6.1 INPUT Interpreter 44
6.2 INPUT_PICK Statement Parameters...............................44
6.2.1 Four Examples Of INPUT_PICK Statements With Their Associated Displays.......................................46
6.3 INPUT_KEY Statement Parameters..............................47
6.3.1 Four Examples Of INPUT_KEY Statements With Associated Displays..48
6.3.2 Limitations And Shortcomings Of INPUT_PICK....50
6.4 Limitations And Shortcommings Of INPUT_KEY............51
6.5 Storage Of Marked Locations For INPUT_PICK...............52
6.6 Glossary...53

Section C7 55
Menu Language 55
7.1 Introduction...55
7.2 Syntax Overview 55
7.3 Initial Syntax Common To Other Interpreters...............56
7.4 Declaration Statement...56
7.5 Box Statement...57
7.6 Header Statement...57
7.7 Choices Statement...57
7.7.1 Button Phrase 58
7.7.2 Size Phrase 58
7.7.3 List Or Text Phrase...59
7.8 Select Statement...60
7.8.1 Return Choice Made Parameter 60
7.8.2 Activate Selects..60
7.8.3 Select Options..61
7.9 Examples 62
7.9.1 Example 1 62
7.9.2 Example 2 62
7.9.3 Example 3 62
7.9.4 Example 4 62
7.9.5 Example 5 63

Section C8 64
Expressions Syntax...64
8.1 Introduction...64

8.2 The Assignment Statement...64
8.3 String Expressions...64
8.4 String Relations..65
8.5 Numeric Expressions...66
8.6 Qualified Data Element...67

Interpretation of Syntax used in this Appendix

The syntax used in this Appendix is a small portion of the Elf langauge designed and developed by Alan Davis of the Division of Educational Research Services. Statements used here are a very small portion of the Elf system. A brief description of the interpretation is given below to enable the reader to understand the syntax of the various interpreters developed.

```
<name>              defines the production called "name"
      is    <a>     calls production "a"
      or    <b>     calls production "b"
;                   ; is end of production marker
```

Note: If production <a> succeeds, execution returns to the production which is called "name". If <a> fails, then the alternate "or" which calls production <b> is executed. If production <b> fails, the production <name> fails and returns the failure to the calling source of <name>.

Some of the parsing notation of Elf are illustrated below:

```
<name2>
      is    "+"           parse for the character +
            [c]...        see note below
      or    "AND"@        parse for AND optionally lower case
      or    1 or more of "abcde"  parse for 1 or more of a,b,c,d, or e
      or    *eoi          test for end of input
      or    *eol          test for end of line
      or    "st" |"op"|   parse for st and optionally op
            /*comment*/         comments between /* and */
            ;
```

Note: [c]... calls production c repeatedly until it returns failure; however the failure is not propogated and the alternate succeeds. The term [c] would call production c once, and would always return a succeed.

Usually a production also contains some semantics, and may contain definitions of abstractions of data structures of various types. For a complete description of Elf see Davis, 1978.

Section C1

Identifiers, Types, and Expressions

1.1 Introduction

In this chapter the various classes of objects within the authoring system that must have an identifier as a label are defined. The scope of identifiers is also defined.

1.2 Identifiers

Identifiers are symbolic names for courses, course_levels, control_modules, content_modules, answer_modules, display_modules, input_modules, menu_modules, routines, functions, restart_points, constants, types, and variables. The terms constant, type,and variable are used in a manner similar to their use in Pascal. The syntax for an identifier is:

```
<identifier>
      is      1 to 31 of ("abcdefghijklmnopqrstuvwxyz"
              "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
              "0123456789"
              "_#$%")  ;
```

There is one restriction; numeric identifiers (functions, variables or constants) may not start with a numeral because of the ambiguity in parsing numeric expressions.

1.2.1 Scope Of Identifiers

The scope of identifiers is static and is similar to that used in Pascal. An identifier defined at one scope level is known at all lower scope levels but maybe redefined at a lower level. For example, an integer variable, say, NumRight, is defined at one level. It is known and may be referenced at that and all lower levels. However, at some lower level it maybe redefined as, say, a LIST OF NUMERIC. Then the outer NumRight may not be referenced at that and lower levels because it has been replaced by the local NumRight. The outer NumRight still exists but may only be referenced when processing returns to the higher level.

Two scope levels are predefined: system and course. Care should be taken by authors in redefining system constants, types, variables, and functions as they would then lose access to them within their course. To assist inexperienced authors system identifiers can be protected from redefiniton by the setting of a switch in the author's registration record.

Within a course an author is free to define the number of and the names of the scope levels in the course. Scope level names may have aliases which in turn define another branch of scope level names. By default three levels are predefined:

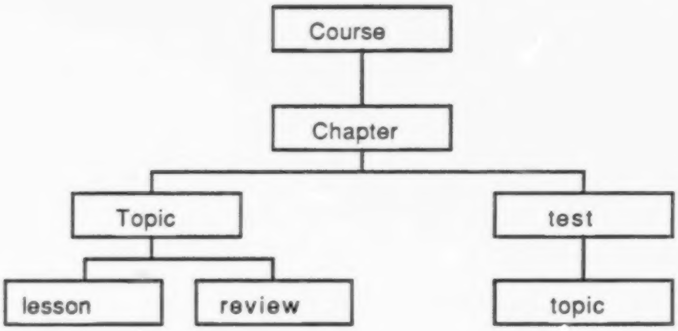


Figure C1 Predefined Levels

These defaults may be altered by an author. In the default case a course is made up of one or more chapters, each chapter of one or more topics or tests, each topic of one or more lessons or reviews, and each test of one or more topics. Notice that the scope level of a topic depends on whether it is part of a chapter or part of a test.

The levels of a course may not easily be changed (though the level names may be) once construction of a course begins, so an author should carefully consider his course level structure before beginning authoring the course. A course level system can always be extended by adding a level to the lowest level. It is also easy to insert a new course level, thus forcing the previous course level to be one instant of a new second level. For example, an author begins defining a course with just two levels, the course level and a lesson level. That is, his course is made up of one or more lessons. He then decides to insert a new course level thus forcing the old course level to be one instant of a new second level that he has named the chapter level. Now his course has three levels: course, chapter, and lesson. Later he decides to add a level below the lesson level and name it the part level. Now his course has four levels: course, chapter, lesson, and part.

An instance of a level is associated with one control module and any number of other module types and data records. Using the default levels for example, the course control module could invoke any content module defined at the course or system level, could reference any routine, function, constant, type or variable defined at the course or system level, and could invoke any chapter control module. In the same way each chapter control module could invoke any content module defined for that chapter or at the course or system level, could reference any routine, function, constant, type or variable defined for that chapter or at the course or system level, and could invoke any topic or test control module that had been attached to that chapter. Each topic control module could invoke any content module defined for that topic or for the chapter that invoked it or at the course or system level, could reference any routine, function, constant, type or variable defined for that topic or for the chapter that invoked it or at the course or system level, and could invoke any lesson or review control module attached to that topic. Display, input, answer, and menu modules may be considered as specialized routines available to content modules only.

1.2.2 Case Sensitivity Of Identifiers

System level identifiers are case insensitive. When referenced the author may type the alphabetics in either case. When listed all system identifiers are in upper case.

At the time a course is set up for an author he must decide if his identifiers should be case sensitive or not. If they are case sensitive, whenever they are referenced the author must type the identifier in the correct case, i.e.,'NumRight' would be a different identifier than 'Numright'.And, of course, they will be listed in upper and lower case. However, if the author decides that his identifiers will be case insensitive, then when referenced they may be typed in either or both cases. The author may decide at anytime how his course identifiers should appear in listings. He may choose uppercase, lowercase, or as typed in when defined. He may also make this choice for all classes of identifiers or for specified classes. The defaults are: variables in lower case, types and constants in upper case, and all others as typed in when defined.

1.3 Routines & Functions

Routines may be called by control modules, content modules, routines, and functions. Content modules may also call other content modules as well as display, input, answer, and menu modules. Functions may be called from within an expression of the same type as the function. A function is a subroutine that returns an explicit result of some defined type. A system_routine is a call to a routine written in Elf and whose name is prefixed by &. Routines, system_routines, and functions may take parameters of both call by value and call by reference and may be recursively entered. Formal parameters may provide for default values or default references.

Routines and functions are created in the same way as control modules. They may define local constants and variables, may reference any routine, function, constant, type, or variable within their scope, and may invoke any content module within their scope. For example a topic control module may reference a routine defined at the course level but that routine could not reference any identifiers below the course level unless they were passed to it by reference because of the scope rules. The scope of local constants and variables is local to the routine and are unknown to locally referenced routines unless passed to them by reference. Routines are defined in association with a particular control module.

System_routines are only defined at the system level. Content, display, input, answer, and menu modules may take parameters of both call by value and call by reference and may define local constants and variables.

1.4 Restart_points & Control Modes

A restart_point automatically generates a course_control mode or a student_control mode restart record in the student's restart record file. This restart record contains all the information the system needs to properly restart the student at that point of the course. A student may be in one of two modes: course_control mode or student_control mode. When in course_control mode, the student's flow through the course is completely controlled by the control component and the student model component.

If the student interrupts this normal course flow to back_up, review, or brouse, then he enters student_control mode that may affect performance recordings, module access, and exam entry. If he signs off in student_control mode he will be restarted in student_control mode using the student_control mode restart record. When the student selects to leave student_control mode his course_control mode restart record is used to restart him at the last restart_point passed while in course_control mode. If a student signs off while in course_control mode he will be restarted in this mode using the course_control mode restartrecord.

An instructor may decide which, if any, of the student_control sub-modes are available to the student. The back_up sub-mode allows the student to back up to previous displays. The instructor may limit the number of previous displays the student may back up through. The system may also have to place some upper limit as well. The review sub-mode allows the student to select for review any unlocked portion of the course that has been marked in his records as having been completed. The brouse sub-mode permits the student to select for review any unlocked portion of the course. The instructor may lock portions of the course, such as exam modules, to these last two sub-modes. The instructor also has control over permittingaccess to certain of the brouse sub-modes features, such as access to correct answers, movement without responding to questions, etc.

Restart_points are both implicit and explicit. Entry into any control module is an implicit restart_point. The name of that control module is the identifier for that restart_point. Within any control or content module an author may explicitly insert a restart_point. The label for that explicit restart_point is appended to the control module name (with an intervening dot) to become the identifier for that restart_point.

1.5 Types, Variables, & Constants

Identifiers for types, variables, and constants may be declared in any control module. Local variables and constants may also be declared in other modules, routine sand functions.

Identifiers for types, variables, and constants may be declared at any point in the authoring process simply by the author requesting the declaration mode. This may be done automatically if the author references an identifier that is undeclared. Declarations at any scope may be created, viewed, or modified. The syntax diagrams in this section reflect the non-interactive declaration and listing forms.

1.5.1 Types

Data are characterized by their type. Type defines both the range of values the data may take, their internal representation, and the way data may be referenced.

```
<type_declaration>
        is      "TYPES" <type_definition>
                 [more_type_definitions]...  ;

<more_type_definitions>
        is      ";" <type_definition> ;

<type_definition>
        is      <identifier> "=" <type> ;

<type>
        is      <simple_type>
        or      <structured_type> ;

<simple_type>
        is      <numeric_type>
        or      <string_type> ;

<numeric_type>
```

```
        is      <numeric_type_identifier>
        or      "NUMERIC" [initial_numeric_value] ;

<initial_numeric_value>
        is      ":=" <numeric_expression> ;

            /* Implemented as the Elf INTEGER type but represents a fixed point decimal
            number with three decimal places.  A numeric is stored as an integer equal to
            the numeric value time 1000.  */

<string_type>
        is      <string_type_identifier>
        or      "STRING" "OF" <max_string_length>
                [initial_string_value]  ;

<initial_string_value>
        is      ":=" <string_expression> ;

<max_string_length>
        is      <numeric_expression>
                /* range 1 to 32767 */ ;

<structured_type>
        is      <list_type>
        or      <record_type> ;

<list_type>
        is      <list_type_identifier>
        or      "LIST" "[" <max_list_elements> "]" "OF"
                <type> ;

<max_list_elements>
        is      <numeric_expression> ;

<record_type>
        is      <record_type_identifier>
        or      "RECORD" <field>
                [more_fields]... "END_RECORD" ;

<more_fields>
        is      ";" <field> ;

<field>
        is      <field_identifier>
                ":" <type> ;
```

1.5.1.1 System Types

In addition to the primitive and structured types given above, the system may define other types. Below are some suggested predefined system types:

/* SYSTEM */ TYPES

```
POINTER = RECORD
        R : NUMERIC
        C : NUMERIC
END_RECORD ;

MENU = LIST [30] OF POINTER ;

MENU_SET = LIST [30] OF MENU
```

1.5.2 Variables

Variables are named references to locations in memory which may contain values pertaining to some type. These values may change during program execution. Variables must be declared as belonging to some type before being referenced. Space allocation is done at execution time on entering the module, subroutine, or function where the variable is declared. Variable space is deallocated on exit from the module, subroutine, or function where it was declared. A variable may not change its type during execution.

```
<variable_declaration>
        is      "VARIABLES" <variable_definition>
                  [more_variable_definitions]...  ;

<more_variable_definitions>
        is      ";" <variable_definition> ;

<variable_definition>
        is      <variable_identifier>
                ":" <type> ;
```

1.5.2.1 Dynamic Variables

Variables of POINTER type may have space allocated to them dynamically at execution time by a POINT statement.

```
<point_statement>
        is      "POINT" <pointer_variable>
                 "TO" <qualified_data_element> ;

<pointer_variable>
        is      <identifier> ;
```

1.5.3 Constants

Constants are named data items whose value is established at author time and cannot be changed at execution time.

```
<constant_declaration>
        is      "CONSTANTS" <constant_definition>
                  [more_constant_definitions]...  ;

<more_constant_definitions>
```

```
        is      ";" <constant_definition> ;

<constant_definition>
        is      <identifier> "=" <constant_value> ;

<constant_value>
        is      [sign] <numeric_value>
        or      <boolean_constant>
        or      <string_value>
        or      <expression> ;

<sign>
        is      "+"
        or      "-" ;

<numeric_value>
        is      *integer /* <= 2147482 */
                [fraction_part]  ;

<fraction_part>
        is      "." *integer /* <= 999 */ ;

<boolean_constant>
        is      "TRUE"
        or      "FALSE"
        or      "YES"
        or      "NO"
        or      "ON"
        or      "OFF" ;

<string_value>
                is      """" 1 or more of anything """" ;
            /* A string_value is enclosed by a quote character on each end that is not part
            of the string.  Two consecutive quote characters within a string_value
            represent one quote character and take only one position in the string.  */
```

1.5.3.1 System Constants

There will be a number of system constants pre-declared. A partial list follows:

1. MAXINT - the largest positive integer supported by the system (2147482)
2. MININT - the lowest negative integer support by the system (-2147482)
3. INFINITY - the largest unsigned real number supported by the system (2147482.999)
4. EPSILON - the smallest unsigned fractional real number supported by the system; a smaller number would be represented by 0 (0.001)

1.6 Expressions

1.6.1 Numeric Expressions

An expression that results in a numeric value is a numeric expression. Boolean expressions are considered as numeric expressions because boolean values are automatically converted to 1 (TRUE) and 0 (FALSE). Numeric expressions are written as standard algebraic infix notation and are evaluated according to the following precedence rules:

1. ( ) - parenthetical expressions
2. unary minus
3. * / DIV MOD
4. + -
5. = <> < > <= >= IN
6. NOT ~
7. AND &
8. OR | XOR

DIV - returns the integer quotient after division
- operands are of numeric type

MOD - returns the remainder after integer division
- may have real result if either operand is real

IN - returns
    a) index (+1) of a substring in a string operands are STRING
    b) 0 if substring not in a string

= <> < > <= >= may compare string operands and operands of numeric type.

1.6.1.1 Numeric Conversion Rules

In an integer expression, a boolean value is converted to 1 (TRUE) or 0 (FALSE). In a real expression, an integer or boolean value is converted to a real value. In a boolean expression, a numeric value of 0 is converted to FALSE and any other numeric value is converted to TRUE.

1.6.2 String Expressions

An expression that results in a string value is a string expression.

```
<string_expression>
        is      <string_term>
                [more_string_terms]...  ;

<more_string_terms>
        is      "+" <string_term>
                /* + means string concatination */
        or      "&" <string_term> ;
                /* & means concatinate with space between */

<string_term>
        is      <string_factor>
                [string_multiplier]  ;
```

```
<string_multiplier>
        is      "*" <numeric_expression> ;

<string_factor>
        is      <string_value>
        or       "(" <string_expression> ")"
        or      <qualified_data_element>
        or       <string_function> ;

<string_value>
        is      /* defined above */ ;

<qualified_data_element>
        is      /* defined below and must reference a string */ ;

<string_function>
        is       /* a function that returns a string */ ;
```

1.6.3 Operands

```
<expression_operand>
        is     <constant_value>
        or      <function_call>
        or     <qualified_data_element>
        or      "(" <expression> ")" ;

<constant_value>
        is      /* defined above */ ;

<function_call>
        is       <function_identifier>
                 [actual_parameters]  ;

<actual_parameters>
        is      "(" <actual_parameter>
                 [more_actual_parameters]...  ")"  ;

<more_actual_parameters>
        is      "," <actual_parameter> ;

<actual_parameter>
        is     <expression>
        or      <qualified_data_element> ;

            /* Actual reference parameters must be matched by a qualified data element of
            the matching formal reference parameter's type.  */

<qualified_data_element>
        is       <variable_identifier>
                 [list_index]...
                 [field_reference]...
                 [subpart_specification]  ;
```

```
<field_reference>
        is      "." <field_identifier>
                [list_index]...  ;

<list_index>
        is      "[" <index_value:>
                [more_index_values]...  "]"  ;

<more_index_values>
        is      "," <index_value> ;

<index_value>
        is      <numeric_expression> ;

<subpart_specification>
        is      /* Same form as in Elf; applies to types STRING; values may be
                <numeric_expression> */ ;
```

Section C2

Control Language
Syntax

2.1 Introduction

The ERAS control language is used to define controlmodules, sub-routines (usually just called routines), and functions. It is specified as a structured procedure oriented language modeled on Pascal and COMAL (COMonAlgorithmic Language). There is one control module at each scope level in a course and it is used to direct the flow ofthe student through the content modules of that level plus the invokation of control modules at the next lowest level.Entry into a course is through the course control module. Routines and functions allow for the definition of algorithmic procedures. Routines are invoked by a ROUTINE statement. Functions return a STRING or NUMERIC explicit result and are call from within <expression>'s.

2.2 Syntax

```
<eras_process>
        is      <control_process>
        or      <routine_process>
        or      <function_process> ;

<control_process>
        is      "CONTROL" <identifier> <eol>
                [parameter]...
                [local_constant]...
                [local_variable]...
                [statement]...
                "END_CONTROL"
                <identifier> <eol> ;

<routine_process>
        is      "ROUTINE" <identifier> <eol>
                [parameter]...
                [local_constant]...
                [local_variable]...
                [statement]...
                "END_ROUTINE"
                <identifier> <eol> ;


<function_process>
        is      "FUNCTION" <identifier> ":" <function_type>
                <eol>[parameter]...
                [local_constant]...
                [local_variable]...
                [statement]...
                "END_FUNCTION" <identifier> <eol> ;
```

```
<function_type>
        is    "NUMERIC"
        or    "STRING" ;

<parameter>
        is     <value_parameter>
        or     <reference_parameter> ;

<value_parameter>
        is      "VAL" <identifier> ":"
                 <value_type> [default_value] <eol> ;

<value_type>
is     "NUMERIC"
or      "STRING" "OF" <numeric_expression> ;

<default_value>
        is      ":=" <expression> ;

<reference_parameter>
        is      "REF" <identifier> ":"
                 <reference_type> <eol> ;

<reference_type>
        is       "NUMERIC" [default_value_reference]
        or       "STRING" [default_value_reference]
        or       "LIST" [default_reference]
        or      "RECORD" [default_reference]
        or       <type_identifier> [default_reference] ;

<default_value_reference>
        is     <default_value>
        or     <default_reference> ;

<default_reference>
        is      "\" <qualified_data_element> ;

<type_identifier>
        is      <identifier> ;
```

2.2.1 Process Body

```
<local_constant>
        is      "CON" <identifier> "="
        <value> <eol> ;

<local_variable>
        is      "VAR" <identifier> ":"
                 <type> <eol> ;

<type>
```

```
            is      /* See Section 1 */ ;

<statement>
            is      <structure_statement>
            or      <simple_statement> <eol> ;

<simple_statement>
            is      <comment>
            or      <null>
            or      <content_call>
            or      <routine_call>
            or      <system_routine_call>
            or      <control_call>
            or      <elf_grammar_call>
            or      <loop_exit>
            or      <return_from_process>
            or      <pointer>
            or      <assignment> ;

<structure_statement>
            is      <loop_structure>
            or      <while_structure>
            or      <until_structure>
            or      <for_structure>
            or      <if_structure>
            or      <case_structure> ;
```

2.2.2 Simple Statements

```
<comment>
            is      "///"
                    0 or more of anything ;

<null>
            is      "NULL" ;

<loop_exit>
            is      "EXIT" "WHEN"
                    <numeric_expression>
                    /* conditional loop exit */
            or      "EXIT"
                    /* unconditional loop exit */ ;

<return_from_process>
            is      "RETURN" [expression] ;

<pointer>
            is      "POINT" <identifier>
                    "TO" <qualified_data_element> ;
```

2.2.3 Process Calls

```
<content_call>
                /* This will be a call to the content module interpreter */

        is      "CONTENT" <identifier>
                  [actual_parameters]  ;

<routine_call>
        is      "ROUTINE" <identifier>
                  [actual_parameters]  ;

<system_routine_call>
        is        "&"<identifier>
                  [actual_parameters]  ;

<control_call>
        is      "CONTROL" <identifier>
                  [actual_parameters]  ;

<elf_grammar_call>

        /* The user must provide an interface grammar with logical name
        ELF$INTERFACE and a goal <ENTRY> with 16 parameters. The first parameter
        will abstract the current STACK, the second will abstract a string containing the
        desired grammar name, and the third will abstract a string containing the desired
        goal name. Each of the remaining 13 parameters may contain an abstraction of
        type STORE_ELEMENT or may be left NOT_IN_USE. The interface grammar must
        reference the following require file before any local type definitions:
        \REQUIRE ERAS$REQ */

        is      "ELF" <grammar_name>
                  "," <goal_name>
                  [13:elf_grammar_parameters]  ;

<grammar_name>
        is        <identifier> ;

<goal_name>
        is        <identifier> ;

<elf_grammar_parameters>
        is      ","  <qualified_data_element> ;

<actual_parameters>
        is        "(" <actual_parameter>
                  [more_parameters]...  ")"  ;

<more_parameters>
```

```
        is      "," <actual_parameter> ;

<actual_parameter>
        /* If an actual parameter is missing, the formal parameteris checked for a
        predefined default. If no default is defined, an error will occur. */

        is      <actual_value_parameter>
        or      <actual_reference_parameter> ;

<actual_value_parameter>
        is      <expression> ;

<actual_reference_parameter>
        is      <qualified_data_element> ;
```

2.2.4 Assignment Statement

```
<assignment>
        is      <qualified_data_element>
                  <assign_operator>
                  <expression> ;

<assign_operator>
        is      ":=" /* NUMERIC or STRING assignment */
        or      ":+" /* NUMERIC increment and STRING append */
        or      ":-" /* NUMERIC decrement */
        or      ":&" /* STRING append with intervening space */
        or      ":C=" /* Assignment with type conversion */ ;
```

2.2.5 Structure Statements

```
<loop_structure>
        is     "LOOP" <eol>
                  <statement>
                  [statement]...
                  "ENDLOOP" <eol> ;

<while_structure>
        is      "WHILE" <numeric_expression>
                  "DO" <eol>
                  <statement>
                  [statement]...
                  "ENDWHILE" <eol> ;

<until_structure>
        is     "REPEAT" <eol>
                  <statement>
                  [statement]...
                  "UNTIL" <numeric_expression> <eol> ;

<for_structure>
```

```
        is      "FOR" <qualified_data_element>
                 ":=" <numeric_expression>
                 "TO" <numeric_expression>
                 [step]
                 "DO" <eol>
                 <statement>
                 [statement]...
                 "ENDFOR" <qualified_data_element> <eol> ;

<step>
        is      "STEP" <numeric_expression> ;

<if_structure>
        is      "IF" <numeric_expression>
                 "THEN" <eol>
                 <statement>
                 [statement]...
                 [elif_structure]...
                 [else_structure]
                 "ENDIF" <eol> ;

<elif_structure>
        is      "ELIF" <numeric_expression>
                 "THEN" <eol>
                 <statement>
                 [statement]... ;

<else_structure>
        is     "ELSE" <eol>
                 <statement>
                 [statement]... ;


<case_structure>
        is     "CASE" <qualified_data_element> <eol>
                 [when_structure]...
                 [otherwise_structure]
                 "ENDCASE" <qualified_data_element> <eol> ;

<when_structure>
        is     "WHEN" <expression>
                 [more_expressions]...
                 "DO" <eol>
                 <statement>
                 [statement]... ;

<more_expressions>
        is      "," <expression> ;

<otherwise_structure>
        is     "OTHERWISE" <eol>
                 <statement>
                 [statement]... ;
```

```
<eol>
        is      [comment] *eol ;
```

2.2.6 Expression Evaluator

```
<expression>
        is      <string_expression>
        or      <numeric_expression> ;

<value>
        is      <string_value>
        or      <numeric_value>
        or      <boolean_value> ;

<string_value>
        is        """" 0 or more (characters) """"
                  /* the quote character is represented by "" in a string */ ;

<numeric_value>
        is      [sign]
                 *integer /* <= 2147482 */
                 [fraction_part]  ;

<sign>
        is      "+"
        or      "-" ;


<fraction_part>
        is       "." *integer /* <= 999 */ ;

<boolean_value>
        is      "TRUE"
        or      "FALSE"
        or      "YES"
        or      "NO"
        or      "ON"
        or      "OFF" ;
```

2.2.7 String Expression Evaluator

```
<string_expression>
        is      <string_term>
                 [more_string_terms]...  ;

<more_string_terms>
```

```
        is      "+" <string_term> /* concatinate */
        or      "&" <string_term> /* concatinate with space */ ;

<string_term>
        is      <string_factor>
                [string_multiplier]  ;

<string_multiplier>
        is      "*" <numeric_expression> ;

<string_factor>
        is      "(" <string_expression> ")"
        or      <string_value>
        or      "!"<string_function_call>
        or      "&"<system_string_function_call>
        or      <qualified_data_element>
                [subpart_specification]  ;

<string_function_call>
        is      <identifier>
                [actual_parameters]
                [subpart_specification]  ;

<system_string_function_call>
        is      <identifier>
                [actual_parameters]
                [subpart_specification]  ;

<subpart_specification>
        is      "{" [left_offset]
                [right_part]  "}"
                /* $ may be used in any subpart numeric expression; it is a special
                variable that represents the current length of the string */ ;

<left_offset>
        is      <numeric_expression>
                /* defaults to zero */ ;

<right_part>
        is      "^" <right_offset>
        or      "~" <length> /* defaults to ^$-1 */ ;

<right_offset>
        is      <numeric_expression> ;

<length>
        is      <numeric_expression> ;
```

2.2.8 Numeric Expression Evaluator

```
<numeric_expression>
```

```
          is      <logical_term>
                    [more_logical_terms]...  ;

<more_logical_terms>
          is      "OR"@ <logical_term>
          or      "XOR"@ <logical_term> ;

<logical_term>
          is      <logical_factor>
                    [more_logical_factors]...  ;

<more_logical_factors>
          is      "AND"@ <logical_factor> ;

<logical_factor>
          is      [negate] <relation> ;

<negate>
          is     "NOT"@ ;

<relation>
          is       <arithmetic_relation>
          or       <string_relation> ;

<string_relation>
          is      <string_expression>
                    <string_relational_operator>
                    <string_expression>  ;

<string_relational_operator>
          is      "IN"
          or       <relational_operator> ;

<arithmetic_relation>
          is      <formula>
                    [more_formulas]...  ;

<more_formulas>
          is       <relational_operator>
                    <formula> ;

<relational_operator>
          is      "<>"
          or      "<="
          or      "<"
          or      ">"
          or      "=" ;

<formula>
          is       [sign] <arithmetic_expression> ;

<arithmetic_expression>
          is       <term> [more_terms]... ;
```

```
<more_terms>
        is      <sign> <term> ;

<term>
        is      <factor> [more_factors]... ;

<more_factors>
        is      <multiplicative_operator>
                <factor> ;

<multiplicative_operator>
        is      "*"
        or      "/"
        or      "DIV"
        or      "MOD" ;

<factor>
        is      "(" <numeric_expression> ")"
        or      <boolean_value>
        or      <numeric_value>
        or      "!"<numeric_function_call>
        or      "&"<system_numeric_function_call>
        or      <qualified_date_element> ;

<numeric_function_call>
        is      <identifier>
                [actual_parameters]  ;

<system_numeric_function_call>
        is      <identifier>
                [actual_parameters]  ;
```

2.2.9 Qualified Data Element Evaluator

```
<qualified_data_element>
        is      <variable_name> [list_index]...
                [field_reference]...  ;

<field_reference>
        is      "." <field_identifier>
                [list_index]...  ;

<field_identifier>
        is      <identifier> ;

<list_index>
        is      "[" <numeric_expression>
                [more_list_index]...  "]"  ;

<more_list_index>
```

```
        is      "," <numeric_expression> ;

<variable_name>
        is      <identifier> ;

<identifier>
        is      1 to 31 of ("abcdefghijklmnopqrstuvwxyz"
                "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
                "0123456789_#$%")   ;
```

Section C3

Content Language
Syntax

3.1 Introduction

The content language is provided for the structuring of content modules. The presentation of information to the student and the processing of student responses to questions is handled by content modules. A content module presents the content of a small portion of a course in a linear sequence selected by the author. The elements in this sequence may be displays of text and graphics, the posing of questions, and the analysis, clasification, and reprise to the student's response. Intermediate calculations and the calling of subroutines may also be done within a content module. Content modules may be invoked by control modules which control the general flow of a student through a course, and by other content modules.

3.2 Syntax

```
<content_process>

        is    "CONTENT:" <identifier> <eol>
                [parameter]...
                [local_constant]...
                [local_variable]...
                [statement]...
                "END_CONTENT" <identifier> <eol> ;

<parameter>
        is      <value_parameter>
        or      <reference_parameter> ;

<value_parameter>
        is      "VAL" <identifier> ":"
                <value_type> <eol> ;

<value_type>
        is     "NUMERIC"
        or      "STRING" "OF" <numeric_expression> ;
                <reference_parameter>is        "REF" <identifier> ":"
                <reference_type> <eol> ;

<reference_type>
        is     "NUMERIC"
        or     "STRING"
        or      "LIST"
        or     "RECORD"
        or      <type_identifier> ;

<type_identifier>
        is      <identifier> ;
```

3.3 Process Body

```
<local_constant>
        is      "CON" <identifier> "="
                 value> <eol> ;

<value>
        is      /* See Section C2 */ ;

<local_variable>
        is      "VAR" <identifier> ":"
                 <type> <eol> ;

<type>
        is      /* See Section C1 */ ;

<statement>
        is      <simple_statement> <eol> ;

<eol>
        is      [comment] *eol ;

<simple_statement>
        is      <comment>
        or      <pointer>
        or      <assignment>
        or      <routine_call>
        or      <content_call>
        or      <display_call>
        or      <input_call>
        or      <answer_call>
        or      <menu_call> ;
```

3.4 Simple Statements

```
<comment>
        is      "///"
                 0 or more of anything ;

<pointer>
        is      "POINT" <identifier>
                 "TO" <qualified_data_element> ;

<assignment>
        is      <qualified_data_element>
                 <assign_operator>
                 <expression> ;

<assign_operator>
        is      /* See Section C2 */ ;
```

```
<expression>
        is      /* See Section C2 */ ;

<qualified_data_element>
        is      /* See Section C2 */ ;

<identifier>
        is       1 to 31 of ("abcdefghijklmnopqrstuvwxyz"
                 "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
                 "0123456789#$%")   ;
```

4.5 Process Calls

```
<routine_call>/* This is a call to the control language interpreter */
        is      "ROUTINE" <identifier>
                 [actual_parameters]  ;

<content_call>/* This is a call to the content language interpreter */
        is      "CONTENT" <identifier>
                 [actual_parameters]  ;

<display_call>/* This is a call to the display language interpreter */
        is      "DISPLAY"
                 <display_body> ;

<display_body>
        is       <identifier>
                 [actual_parameters]
        or      <eol>
                 [display_statement]...
                 "END_DISPLAY" ;

<display_statement>
        is       /* See Section C4 */ ;

<input_call>/* This is a call to the input language interpreter */
        is      "INPUT"
                 <input_body> ;

<input_body>
        is       <identifier>
                 [actual_parameters]
        or      <eol>
                 [input_statement]...
                 "END_INPUT" ;

<input_statement>
        is       /* See Section C6 */ ;

<answer_call>/* This is a call to the answer language interpreter */
        is      "ANSWER"
                 <answer_body> ;
```

```
<answer_body>
        is      <identifier>
                 [actual_parameters]
        or     <eol>
                 [answer_statement]...
                 "END_ANSWER" ;

<answer_statement>
        is      /* See Section C5 */ ;

<menu_call>/* This is a call to the menu language interpreter */
        is      "MENU"
                 menu_body> ;

<menu_body>
        is      <identifier>
                 [actual_parameters]
        or     <eol>
                 [menu_statement]...
                 "END_MENU" ;

<menu_statement>
        is      /* See Section C7 */ ;

<actual_parameters>
        is      "(" <actual_parameter>
                 [more_parameters]...  ")"  ;

<more_parameters>
        is      "," <actual_parameter> ;

<actual_parameter>
        is      <actual_value_parameter>
        or      <actual_reference_parameter> ;

<actual_value_parameter>
        is      <expression> ;

<actual_reference_parameter>
        is      <qualified_data_element> ;
```

Section C4

Display Language
Syntax

4.1 Introduction

The display language is provided for the structuring of display modules. The display language provides for the presentation of text and graphics information to the student and for the control of audio-visual devices attached to the student's terminal.

Display statements may also appear as lines between a DISPLAY line and an END_DISPLAY line in a CONTENT module or an ANSWER module.

4.2 Syntax

```
<display_process>
        is      "DISPLAY" <identifier> <eol>
        [parameter]...
        [local_constant]...
        [local_variable]...
        [statement]...
        "END_DISPLAY" <identifier> <eol> ;

<parameter>
        is      <value_parameter>
        or      <reference_parameter> ;

<value_parameter>
        is      "VAL" <identifier> ":"
                <value_type> <eol> ;

<value_type>
        is      "NUMERIC"
        or      "STRING" "OF" <numeric_expression> ;

<reference_parameter>
        is      "REF" <identifier> ":"
                <reference_type> <eol> ;

<reference_type>
        is      "NUMERIC"
        or      "STRING"
        or      "LIST"
        or      "RECORD"
        or      <type_identifier> ;

<type_identifier>
        is      <identifier> ;
```

4.3 Process Body

```
<local_constant>
        is      "CON" <identifier> "="
                 <value> <eol> ;

<value>
         is      /* See Section C2 */ ;

<local_variable>
        is      "VAR" <identifier> ":"
                 <type> <eol> ;

<type>
        is      /* See Section C1 */ ;

<statement>
        is     <simple_statement> <eol> ;

<eol>
        is      [comment] *eol ;

<simple_statement>
        is     <comment>
        or       <routine_call>
        or      <pointer>
        or     <assignment>
        or      <textbox>
        or      <drawbox>
        or     <erase>
        or      <wait>
        or      <text>
        or     <deletebox>
        or      <draw> ;
```

4.4 Common Simple Statements

```
<comment>
        is        "///"
                  0 or more of anything ;

<pointer>
        is      "POINT" <identifier>
                 "TO" <qualified_data_element> ;

<assignment>
        is      <qualified_data_element>
                 <assign_operator>
                 <expression> ;

<assign_operator>
        is      /* See Section C2 */ ;
```

```
<expression>
        is      /* See Section C2 */ ;

<qualified_data_element>
        is      /* See Section C2 */ ;

<identifier>
        is      1 to 31 of ("abcdefghijklmnopqrstuvwxyz"
                "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
                "0123456789_#$%")  ;

<routine_call>
        /* This is a call to the control language    interpreter */
        is      "ROUTINE" <identifier>
                [actual_parameters]  ;

<actual_parameters>
        is      "(" <actual_parameter>
                [more_parameters]...  ")"  ;

<more_parameters>
        is      "," <actual_parameter> ;

<actual_parameter>
        is      <actual_value_parameter>
        or      <actual_reference_parameter> ;

<actual_value_parameter>
        is      <expression> ;

<actual_reference_parameter>
        is      <qualified_data_element> ;
```

4.5 Display Simple Statements

Two types of DISPLAY BOXES :

1) textbox : textbox is created to display text. Coordinate system used : row and column system, where row and column start at the top left corner of the box. Values of row and column : integer , starting at 1.

2) drawbox : drawbox is created to display graphics.Coordinate system used: x_y system, where x and y start at the bottom left corner of the box. Values of x and y: integer ,

    a) can be either positive or negative , and

    b) x_left < x_right , y_bottom < y_top .

To create a DISPLAY BOX :

1) textbox : see section 4.5.1 ;
2) drawbox : see section 4.5.3 .

4.5.1 Textbox And Related Statements

```
<textbox>
                    is    "TEXTBOX" <box_name> [options]... <eol>
                            "(" <textbox_parameters> ")" ;
                    /* Initially the DISPLAY INTERPRETER will  automatically create
                    a textbox, where :

                    Size : same as the current screen
                    No. of rows : as defined for the current screen
                    No. of columns : as defined for the current screen
                    Box  identifier: '0'
                    Note : An attempt to delete this box will cause an error.    */

<box_name>
        is      <identifier> ;

<options>
        is     "FRAMED"
                 /* By default, a box will not be framed. */
        or     "LOCAL"
                 /* If a box is declared LOCAL, it will be deleted automatically when the
                 DISPLAY INTERPRETER exits. By default, a box will be declared GLOBAL
                 and exists as long as the current screen exists. */
        or     "TRANSLUCENT" ;
                 /* By default, a box will be opaque. */

<textbox_parameters>
        is     <text_coordinates> ";"
                 <location> ;

<text_coordinates>
        is     "ROWS = " <numeric_expression> ","
                 "COLS = " <numeric_expression> ;

<location>  /* Location of the new box within the screen. */
        is     <coll> "," <colr> ","
                 <rowb> "," <rowt> ;

<coll>   /* column_left */
        is  <numeric_expression> ;

<colr>   /* column_right */
        is  <numeric_expression> ;

<rowb>   /* row_bottom */
        is  <numeric_expression> ;

<rowt>   /* row_top */
        is  <numeric_expression> ;
```

Example for creating textboxes :

```
TEXTBOX alpha FRAMED(ROWS = 15, COLS = 70 ; 5, 75, 20, 5)

                    /* This statement will create a textbox where :
                    1) it is named "alpha",
                    2) it is framed,
                    3) it is global (by default),
                    4) it is opaque (by default),
                    5) no_of_rows : 15,
                    6) no_of_cols : 70,
                    7) coll : 5,
                    8) colr : 75,
                    9) rowb : 20,
                    10) rowt : 5 */

<erase>
        is      "ERASE" [box_name] ;
                /* Defaults to screen */

<deletebox>
        is      "DELETEBOX" <box_name> ;

<wait>
        is      "WAIT" [wait_value] ;
                /* If there is no wait value or if the wait value is less than 1 then display
                the WAIT_MSG in the centre of the bottom screen row and wait for a
                RETURN key press. If wait value is a string expression display the
                resultant string as the wait message and handle as above. If wait value is a
                numeric expression then pause for the duration of the wait value in
                seconds. */

<wait_value>
        is      <numeric_expression>
        or      <string_expression> ;

<text>
        is      "TEXT" [box_name] [location] <eol>
                [text_start]
                [paragraph]... ;

<location>
        is      "AT" [startrow] [startcol] ;

<startrow>
        is      <numeric_expression> ;
<startcol>
        is      "," <numeric_expression> ;
<text_start>
        is      "PAGE" [page_message] <eol>

                /* (1) IF <page_message> present, print the result on the system screen,
                        ELSE print the default PAGE_MSG on the system screen.
                (2) Wait for RETURN to be pressed.
                (3) Erase the box.
                (4) IF <startrow> present, start text on the result row ELSE start text in
                        row 1.
                (5) IF <startcol> present, start text in the result column ELSE start text
                        in column 1. */
        or      "CLEAR" <eol>
                /* DO actions (3) to (5) above. */
        or      "CONTINUE" <eol> ;
                /* (1) IF <startrow> present, start text on the result row ELIF
                current_row = 1, start text on row 1
                ELSE start text on current_row + 1.
                (2) DO action 5 above. */

<paragraph>
        is      "PARAGRAPH" <ptype> <pstyle> <eol>
                [display_attributes_changed]...
                text_line]... ;

<ptype>
        is       "REGULAR" /* normal paragraph */
        or        "LEVEL_1" /* outline level 1 */
        or        "LEVEL_2" /* outline level 2 */
        or        "LEVEL_3" /* outline level 3 */
        or       "LEVEL_0" /* end outline mode */
        or       "LEVEL_CONT" /* a continuation */
                /* paragraph at current level */ ;

<pstyle>
        is      "PRESET"
                /* text_lines are preformatted Elf command sets. */
        or      "FORMAT"
                /* text_lines are formatted at run time because text
                contains \r or \v text commands. */
        or       "AS_IS" ;
                /* text_lines to be displayed as is; \-, \_, \r, and \v are
                ignored. */

<display_attributes_changed>
        is      "LEFT MARGIN" <integer>
        or      "RIGHT MARGIN" <integer>
        or       "INDENT" <integer>

                /* This is a setting for an indent or out-dent on the first line of a
                paragraph or the first level of an outline. An out-dent is a negative
                integer whose absolute value may not exceed the value of LEFT MARGIN.
                INDENT is 0 for block style, positive for normal paragraphs, and negative
                for point-hang. */

        or       "JUSTIFY" <justify_type>
        or      "SPACING" <spacing_type>
                /* within paragraphs */
        or      "SEPARATION" <spacing_type>
                /* between paragraphs */
        or      "TEXT COLOUR" <colour_name>
        or     "DRAW COLOUR" <colour_name>
        or     "TABS" <tab_specification>
        or     "KEEP" <integer>
               /* This attribute only applies to the immediate paragraph. It specifies the
               number of lines that must be remaining in the text box to continue
               outputing text. If there isn't enough lines left, the PAGE_MSG is displayed
               and the system waits for a RETURN key press before clearing the text box
               and continuing the display. */

        or     "OUTLINE"
                " LEVEL 1" <counter> <indent>
                " LEVEL 2" <counter> <indent>
                " LEVEL 3" <counter> <indent> ;

<justify_type>
        is     "LEFT"
        or     "RIGHT"
        or     "CENTRE"
        or     "BOTH" ;

<spacing_type>
        is     "SINGLE"
        or     "DOUBLE"
        or     "TRIPLE"
        or     "QUADRUPLE" ;

<colour_name>
        is     /* the name of one of the supported colours. */ ;

<tab_specification>
        is     /* to be defined later. */ ;

<counter>
        is     <string_of_length_3> ;

               /* The first and last character are the literal characters that preceed and
               follow the level counter. The second character is a symbol for the type of
               counter. The following symbols are supported:

                       D - decimal,
                       l - lowercase letters,
                       L - uppercase letters,
                       r - l/c roman numerals,
                       R - u/c roman numerals.
                       N - numerate
                       B - bullet
                       The default definitions are:
                       " D." ; "(l)" ; "(r)"
                       This would give the following outline form:
                       1.
                               (a)
                                       (i)
                                       (ii)
                                       (iii)
                (b)
        2.
                (a)
                (b)
                (c)
        3.

        A NUMERATE outline format may also be  specified.  This would
        give the following outline form:
        1.
                1.1
                        1.1.1
                        1.1.2
                        1.1.3
                1.2
        2.
                2.1
                2.2
                2.3
        3.                      */

<indent>
        is      <integer> ;

<text_line>
        is      1 or more of anything
                 *eol ;
```

4.5.2 Embedded Text Commands

A text line contains either or both text to be displayed or text commands. Text for display may be any displayable character. A text command is not displayed but is removed from the text and interpreted by either the display language interpreter or the Elf Screen Support system. A text command may appear anywhere in the text to be displayed. It consists of a text command delimiter, "\", and a single character command. Any parameters are enclosed in parentheses. No spaces are allowed in a text command.

The most common Screen Support commands are:

\cx - change colour where x={r|g|y|b|m|c|w}
\\ - display "\" character

The following are the display language commands:

\- - optional end-of-line word splitter
_ - non-line splitting space
\m(<point_variable>) - mark point; the cursor position at this point in the text is p laced in a variable of type POINT.
\r(<text_module>) - recall text module; at this pointinsert the text to be found in the named module of type TEXT. This module is interpreted as lines of text and text commands. An end-of-module will cause a return to the point just after the \r command of the calling process. The \r command may be nested.

\t - tab; move the current cursor position to the nexttab stop. If no tabs are set, go to column 1 of the next row.

\v(<exp>[,<width>[,<type>[,<decimals>]]]) - insert variable data; <exp> is evaluated as either a string or numeric expression with the results inserted into the surrounding text according to the following rules. If <width> is not present or is less than 1 then the results are inserted with as much width or space as needed, otherwise it specifies the exact width and the results are right justfied in this field. If the results won't fit in the field the field is filled with "*"'s. If <type> is not present then <exp> is evaluated as a string expression, otherwise it is evaluated as a numeric expression. If <type> is 0 the numeric result is displayed as a number with the number of decimal places indicated by <decimals>. If <decimals> is not present then the number is rounded to two decimal places. If <type> is greater than 0, the results are interpreted as boolean and are displayed as shown below:

	expression value	
<type>	0	not 0
1	0	1
2	NO	YES
3	OFF	ON
4	FALSE	TRUE
5	no	yes
6	off	on
7	false	true

4.5.3 Drawbox And Related Statements

```
<drawbox>
        is     "DRAWBOX" <box_name> [options]... <eol>
                 "(" <drawbox_parameters> ")" ;
<box_name>
        is      <identifier> ;

<options>
        is     "FRAMED"
                 /* By default, a box will not be framed. */
        or     "LOCAL"
                 /* If a box is declared LOCAL, it will be delete automatically when the
                 DISPLAY INTERPRETER exits. By default, a box will be declared GLOBAL
                 and exists as long as the current screen exists. */
        or     "TRANSLUCENT" ;
                 /* By default, a box will be opaque. */

<drawbox_parameters>
        is      <draw_coordinates> ";"
                 <location> ;
<draw_coordinates>
        is      <xl> "," <xr> "," <yb> "," <yt> ;
```

```
<xl>
        is  <numeric_expression> ;

<xr>
        is  <numeric_expression> ;

<yb>
        is  <numeric_expression> ;

<yt>
        is  <numeric_expression> ;

<location>
                /* Location of the new box within the screen. */
        is      <coll> "," <colr> ","
                 <rowb> "," <rowt> ;

<coll>   /* column_left */
        is  <numeric_expression> ;

<colr>   /* column_right */
        is  <numeric_expression> ;

<rowb>   /* row_bottom */
        is  <numeric_expression> ;

<rowt>   /* row_top */
        is  <numeric_expression> ;
```

Example for creating drawboxes :

```
DRAWBOX beta LOCAL TRANSLUCENT (-300, 300, -300, 300 ; 1, 60, 12, 1)

                /* This statement will create a drawbox where :
                1) it is named "beta",
                2) it is not framed (by default),
                3) it is local,
                4) it is translucent,
                5) xl : -300,
                6) xr : 300 ,
                7) yb : -300,
                8) yt : 300;
                9) coll : 1,
                10) colr : 60,
                11) rowb : 12,
                12) rowt : 1 */
```

```
<erase>
        is    "ERASE" <box_name> ;

<deletebox>
        is    "DELETEBOX" <box_name> ;
<draw>
```

```
        is     "DRAW" <box_name> <eol>
                 [draw_start]
                 [draw_line]...    ;

<draw_start>
        is     "PAGE" [page_message] <eol>
                 /* (1) IF <page_message> present, print the result on the system screen,
                        ELSE print the default PAGE_MSG on the system screen.
                 (2) Wait for RETURN to be pressed.
                 (3) Erase the box.
                 (4) Display the drawing. */
        or     "CLEAR" <eol>
                 /* DO actions (3) and (4). */
        or      "CONTINUE" <eol> ; /* DO action (4). */

<draw_line>
        is      1 or more of anything *eol ;
```

4.5.4 Embedded Draw Commands

The embedded draw commands are:

\g(<graph_module>) - recall graph module; This module contains 2 columns of integers corresponding to the x and y coordinates.e.g.

```
350   0    /* The 1st column */
312   3    /* corresponds to */
274   9    /* x_coordinate   */
236  24    /* and 2nd column */
198  56    /* corresponds to */
160 110     /* y_coordinate. */
```

At this point plot the graph to be found in the named module of type GRAPH. An end-of-module will cause a return to the point just after the \g command of the calling process.

\r(<draw_module>) - recall draw module; at this point insert the drawing commands to be found in the named module of type DRAW. The \r command may be nested.

Section C5

Answer Language Syntax

5.1 Introduction

The ERAS answer language is provided for the structuring of answer modules. The answer language provides the means to accept a student response, to classify the response for judging, and to provide feedback to the student based on the classified response. Answer statements may also appear as lines between an ANSWER line and an END_ANSWER line in a CONTENT module.

5.2 Syntax

An ANSWER module has the following features:

- There must be exactly one INPUT statement in every ANSWER
- The rest of the ANSWER is composed of any number of <simple_statements > productions (see Section C3) and acons.
- an acon, or answer condition statement, is a specialized control structure unique to the ANSWER language.

```
<answer_process>
        is      "ANSWER:" <identifier> <eol>
                 [parameter]...
                 [local_constant]...
                 [local_variable]...
                 [answer_statement]...
                 <input_statement>  /* see Section C6 */
                 [answer_statement]...
                 "END_ANSWER" <identifier> <eol>;

<parameter>
        is      <value_parameter>
        or      <reference_parameter> ;

<value_parameter>
        is      "VAL" <identifier> ":"
                 <value_type> <eol> ;

<value_type>
        is      "NUMERIC"
        or      "STRING" "OF" <numeric_expression> ;

<reference_parameter>
        is      "REF" <identifier> ":"
                 <reference_type> <eol> ;
```

```
<reference_type>
        is     "NUMERIC"
        or     "STRING"
        or      "LIST"
        or     "RECORD"
        or       <type_identifier> ;

<type_identifier>
        is       <identifier> ;

<local_constant>
        is       "CON" <identifier> "="
                  <value> <eol> ;

<value>
         is      /* See Section 2 */ ;

<local_variable>
        is       "VAR" <identifier> ":"
                  <type> <eol> ;

<type>
C1  */ ;
<answer_statement>
        is     <acon>
        or       <simple_statement> /* see Section C2 */  ;

<acon>
        is      <category_name> <numeric_expression> <eol>
                  <acon_action_part> ;

<category_name>
        is       <identifier> ;

<acon_action_part>
        is       [acon_action]... ;

<acon_action>
        is     "DO" <eol>
                  [simple_statement]...
                  <acon_branch> ;

<acon_branch>
        is       "RETRY" /* goto INPUT reference */
        or       "EXIT"  /* exit ANSWER           */
        or       "CONT"  /* goto next statement  */ ;
```

5.3 Categories

A category name identifies a variable of type CATEGORY. CATEGORY is a predefined record type consisting of:

- STATUS -- an integer set to 0 at entry to current scope
- COUNTER -- an integer set to 0 at entry to current scope

When the answer condition statement is executed the STATUS is set to the value of the numeric_expression. If the expression is true the COUNTER is incremented by 1 and the action_part is executed. If the expression is not true the COUNTER is not incremented, the action_part is not executed, and the acon passes control to the following statement. There is only one predefined category. Its name is IF.

5.4 Acon Action

Every acon has its own pointer which points to the next action to be executed. The pointer is initialized to the first action at entry to the scope from which the acon is referenced. When an action_part is executed the pointer is updated to point to the next action. When all the actions have been used up the pointer remains at the last action. Example:

```
ANSWER:quest3
REF num_right : NUMERIC
INPUT
OK &scan("Beijing") OR &scan("Peking")
DO
DISPLAY right3
num_right  :+1
EXIT
UNREC TRUE
DO
DISPLAY unrec1
RETRY
DO
DISPLAY unrec2
RETRY
DO
DISPLAY unrec3
RETRY
END_ANSWER quest3
```

5.5 System Routines, Functions, And Response Analysis

For the near future, the only special support for response analysis will be in the form of specialized functions. It is hoped that this approach will leave the door open for future response analysis systems. Actually, numeric expressions can be a fairly powerful facility for response analysis if appropriate system functions are made available. If character positions are returned by string search functions, the response can even be parsed by a crude kind of grammar.

The system routines and functions for response analysis are given here. Some common parameters are:

- keys - A string containing characters which are searched for.

- count - A numeric specifying the maximum number of operations. If it is 0, all possible operations are performed. It returns with the actual number of operations performed.
- source - A string which is to be searched and operated on.
- mask - A string which has the same length as the source containing all zeros except that it has 'FF's in positions corresponding to where key characters have been found in the source.

System routines used mainly to edit the response are:

```
&character_delete ( keys | count, mask, source )
        VAL keys : STRING
        REF count : NUMERIC := 0
        REF mask : STRING
        REF source : STRING \Response
```

Deletes all key characters from source. Returns the number deleted in count and their positions in mask.

```
&character_delete_extra ( keys | count, mask, source )
        VAL keys : STRING
        REF count : NUMERIC := 0
        REF mask : STRING
        REF source : STRING \Response
```

Deletes key characters from source which are sitting immediately to the right of a character of the same type. Returns number of characters deleted in count, and their positions in mask.

```
&character_purge ( keys | count, mask, source )
        VAL keys : STRING
        REF count : NUMERIC := 0
        REF mask : STRING
        REF source : STRING \Response
```

Deletes same characters as character_delete_extra plus ALL key characters appearing at the beginning and end of the source string (before the first and after the last non-key character). Returns number of characters deleted in count and their positions in mask.

```
&character_search ( keys | count, mask, source )
        VAL keys : STRING
        REF count : NUMERIC := 0
        REF mask : STRING
        REF source : STRING \Response
```

Searches the source for key characters. Returns the number found in count and their positions in mask.

&character_substitute (char, keys | count, mask, source)
VAL char : STRING
VAL keys : STRING
REF count : NUMERIC := 0
REF mask : STRING
REF source : STRING \Response

char contains a single character which is substituted for all occurrences of key characters in the source. Returns the number substituted in count and their positions in mask.

&substitute (key, new | count, source)
VAL key : STRING
VAL new : STRING
REF count : NUMERIC := 0
REF source : STRING \Response

The source string is searched for occurrences of key string. If found they are replaced by new. The number of occurrences found is returned in count.

System functions which would appear mainly in the numeric expression part of acons are:

&scan (key | case, start, end, source)
VAL key : STRING
REF case : NUMERIC := 0
REF start : NUMERIC := 0
REF end : NUMERIC := (length of source)
VAL source : STRING \Response

source is searched starting at position start for the first occurrence of key before position end. case is a numeric containing 0/1 indicating whether the search is insensitive/sensitive to case. &scan returns 1 if the key is found and 0 otherwise. If the key is found, start will contain its starting position and end will contain the position of its last character plus one.

&pattern_scan (pattern | case, start, end, source)
VAL pattern : STRING
VAL case : NUMERIC := 0
REF start : NUMERIC := 0
REF end : NUMERIC := (length of source)
VAL source : STRING \Response

Works the same as &scan except that the pattern must match the entire portion of source from start to end-1, and unlike the key in &scan the pattern may have embedded operators for pattern matching.

The operators are:
- * any substring (including null)
- + any substring (excluding null)

@	any substring of length one or zero
#	any substring of length one
|s|	substring s is optional but may not be partially present
_	spaces or beginning or end of string

For example, "_al|ber|ta_" would match "alberta" and "alta", but not "albta" nor "call rita". The pattern ""*event@al@y*"" would match "eventually", "eventially", "eventaly", but also "Prevent ally muggings!". This unfortunate match could be eliminated by changing the pattern to ""*_event@al@y_*"". Notice that the pattern specifies a kind of grammar which, however limited, implies backtracking.

The operators *, +, and _ will often appear at the beginning of a pattern, but in these cases it will not be very useful to return the position they match in the start parameter. Therefore start will contain the position matching the first operator (or character) excluding these three. Similarly, end will contain the position matching the last operator (or character) excluding these three.

The operator characters (*,+,@,#,_,|) are treated as literals when preceded by a backslash (e.g. \@). Two adjacent backslashes will be treated as a single literal backslash.

```
&spell ( key | case, threshold, start, end, markup_string, source)
        VAL key : STRING
        VAL case : NUMERIC := 0
        VAL threshold : NUMERIC := 75
        REF start : NUMERIC := 0
        REF end : NUMERIC := (length of source)
        REF markup_string : STRING
        VAL source : STRING \Response
```

The source is searched starting at position start for the first edit distance match of key before position end. Unlike &scan, &spell will only match with a substring delimited by blanks. The threshold is a value between 0 and 100 and defaults to 75. A high threshold reduces the probability of a match, and a low threshold increases the probability of a match. If a match is found, markup_string contains edit codes for passing to a markup procedure. This function returns 1 if a match is found and 0 otherwise. If a match is found, start will contain its starting position and end its ending position. Both the key and words in the source will be truncated to 32 characters before comparison.

```
&similar ( key | case, markup_string, source)
        VAL key : STRING
        VAL case : NUMERIC := 0
        REF markup_string : STRING
        VAL source : STRING \Response
```

This function returns a measure of the similarity between the key and the source strings. The returned value ranges between 0, indicating no similarity, and 100, indicating that the strings are identical. It is normalized such that a single difference between short strings results in a lower similarity than would the same difference between long strings. The author can specify a cost for case conversion through the case parameter. Costs may range from 0 to 10, but a low cost (such as 1) is recommended.

The markup_string will contain edit codes showing how the source can be corrected. It can be passed to a markup routine or used by the author to further analyse the response.

```
&markup( response, markup_string)
        REF response : STRING
        REF markup_string : STRING
```

This is a routine (not a function) which complements &spell. &markup alters the response and the markup_string such that when displayed together they form a feedback message showing the student how to correct his response. Typically this would be used by first calling &spell then calling the &markup passing it the markup_string returned by &spell.

```
&num ( n | low, high, start, end, source) REF n : NUMERIC
        VAL low : NUMERIC := negative infinity
        VAL high : NUMERIC := positive infinity
        REF start : NUMERIC := 0
        REF end : NUMERIC := (length of source)
        REF source : STRING \Response
```

The source is searched starting at position start for the first occurrence of a numeric before position end which falls within the range specified by the bounds low and high. &num returns 1 if a numeric is found and 0 otherwise. If a numeric is found start will contain its starting position and end its ending position plus one, and n will contain its numeric value. Syntax for a numeric is:

[sign] <digit> [digit]... [point] [digit]...
or
[sign] <point> <digit> [digit]...

```
&nums (nlist | start, end, source)
        REF nlist : LIST of NUMERIC
        REF start : NUMERIC := 0
        REF end : NUMERIC := (length of source)
        REF source : STRING \Response
```

The source is searched starting at position start for all numerics which occur before position end. The numerics are returned in nlist. This function returns the number of numerics found. start is set to the beginning of the first numeric. end is set to the last position of the last numeric.

Section C6

Input Language Syntax

6.1 INPUT Interpreter

The input interpreter provides the facility for a keyboard response to be made to a displayed question. It is to be used in conjunction with two other ERAS components, the DISPLAY and ANSWER interpreters, a typical sequence of use being DISPLAY-INPUT-ANSWER. After a question is presented with DISPLAY, INPUT is used to permit the user to respond. ANSWER can then be used to analyse the response.

INPUT permits responses to either a multiple choice question (a menu) or to a question requiring a typed response. Within the interpreter these two input types are called INPUT_PICK and INPUT_KEY respectively.

INPUT_PICK accommodates multiple choice displays configured vertically or horizontally, and two dimensional multiple choice displays. The appropriate pair of arrow keys (left/right or up/down) is used to move the pointer over the single dimensional display of choices; both sets of arrow keys are enabled for a two dimensional display. If the choices are displayed with identifiers (the numbers 1-9, the letters a-z, or the upper case letters A-Z) then these keys can be used to move the pointer directly to a choice. The RETURN key is used to select the choice at which the pointer sits.

On multiple selection INPUT_PICKs the DELETE key is also enabled to permit deletion of a earlier selection; RETURN must therefore be pressed at least as many times as the required number of selections although the user can override this. When a choice is selected, a star is displayed in its marked location to aid the user. Deletion of a selection allows for a replacement, and duplicate selections are ignored.

INPUT_KEY allows either single or multiple row typed responses. For both, the BACKSPACE key can be used to erase everything typed before RETURN is pressed and to restart the response (but not the timer). For multiple row responses the LINEFEED key can be used to move the cursor to the start of the next row. Overflow will otherwise cause it to move there automatically. RETURN terminates the response. Typed input appears in the specified box which was previously defined in the ware file. INPUT_KEY can optionally display hyphens, asterisks or an underscore in the box to indicate the maximum input length. If the start location is defaulted INPUT_KEY uses the next available position as determined by DISPLAY. INPUT_KEY uses the boundaries of the input area to set the allowed length of input, if that is defaulted. The left and right arrow keys are used to move the cursor over some previously typed characters to modify it before entry.

6.2 INPUT_PICK Statement Parameters

INPUT_PICK has 6 parameters. The first three are required, while the last three are optional. They must appear in the order indicated, each separated by a comma. Intervening spaces are ignored.

1. box number:

-the number of the box defined in the ware and within which the marked locations, and usually their associated choices, occur

2. name of points vector:

   -the name of the vector in which the marked locations are saved

   -passed to INPUT_PICK so marked locations can be retrieved

   -appears in the ware with the choice. Two examples are:

   mc(PLAYER[3]) Plante
   mc(COUNTRY[4,2]) Peru

   Here PLAYER and COUNTRY are vector names.

3. number of choices:

   - one number specifying the number of displayed choices of a one dimensional display, or two numbers separated by a semicolon specifying the number of rows and columns of a two dimensional display. For example, 5;4 represents a two dimensional display of 5 rows by 4 columns.

   -all numbers must be in the range 2-10

4. additional available keys:

   -specifies the set of keys to be made available, in addition to the arrow keys, in order to allow pointer movement directly to a choice

   -to be specified only if identifiers are displayed with the choices

   -three options are available:

   NUMBERS enables the numeric keys 1 to 9
   LETTERS_L_CASE enables the lower case letters a to z
   LETTERS_U_CASE enables the upper case letters A to Z

   -for any one of the three key sets, only as many keys are enabled as there are choices displayed

   -a maximum of 9 choices with NUMBERS and 26 with either LETTERS_L_CASE or LETTERS_U_CASE is possible.

   -defaults to none

5. maximum time:

   -specifies the time allowed before INPUT_PICK is terminated

   -on multiple selection INPUT_PICKs if expiry occurs then the selections made to that time will be returned

-expressed in seconds

-defaults to system time

6. number of selections:
   -one number specifying the maximum number of valid choices which can be selected for one execution of INPUT_PICK

   -no termination of INPUT_PICK until the number specified have been selected

   -maximum of 10
   -defaults to 1

6.2.1 Four Examples Of INPUT_PICK Statements With Their Associated Displays

Example 1

INPUT_PICK, 3, PLAYER, 8, , 60,

box number: 3
name of points vector: PLAYER
number of choices presented: 8
additional available keys: defaulted to none
maximum time: 60 seconds
number of selections: defaulted to 1

Example 2

INPUT_PICK, 2, PLAYER, 8, NUMBERS, 120, 3

box number: 2
name of points vector: PLAYER
number of choices presented: 8
additional available keys: number keys 1-8
maximum time: 120 seconds
number of selections: 3

Example 3

INPUT_PICK, 4, COUNTRY, 5;4, , 60,

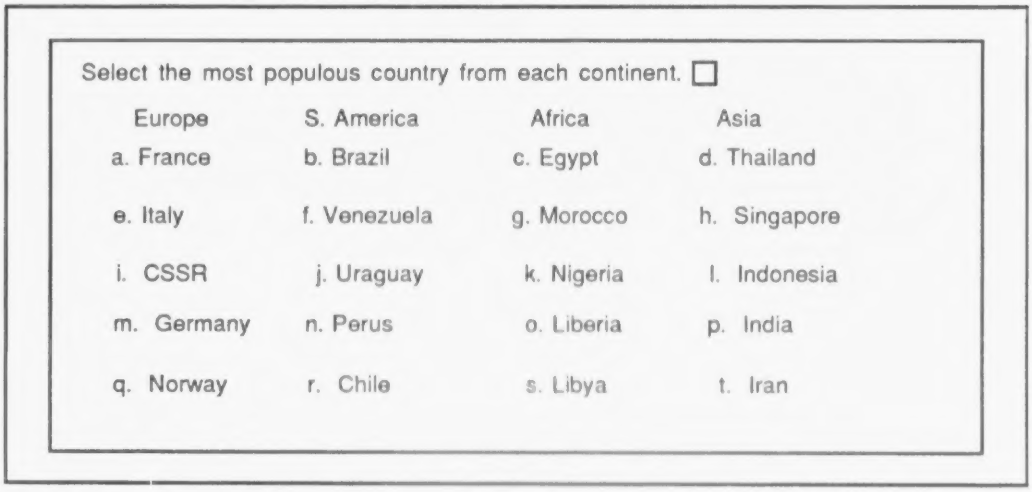
box number: 4
name of points vector: COUNTRY
number of choices presented: 20 in 5 rows of 4 columns
additional available keys: defaulted to none maximum
time: 60 seconds
number of selections: defaulted to 1

Example 4

INPUT_PICK, 5, COUNTRY, 5;4, LETTERS_L_CASE, 180, 5

box number: 5
name of points vector: COUNTRY
number of choices presented: 20 in 5 rows of 4 columns
additional available keys: letter keys a-t
maximum time: 80 seconds
number of selections: 4

See Figure C6.1 below



Select the most populous country from each continent. ☐

Europe	S. America	Africa	Asia
a. France	b. Brazil	c. Egypt	d. Thailand
e. Italy	f. Venezuela	g. Morocco	h. Singapore
i. CSSR	j. Uraguay	k. Nigeria	l. Indonesia
m. Germany	n. Perus	o. Liberia	p. India
q. Norway	r. Chile	s. Libya	t. Iran

Figure C6.1 Example 4 of Input Pick

6.3 INPUT_KEY Statement Parameters

INPUT_KEY has 6 parameters, all but the first being optional. They are to appear in the following order, each separated by a comma with optional spaces.

1. box number:
    - the number of the box in which the keyed input is to appear
2. start row:
    - the row number at which the cursor is to first appear
    - range is determined by the box definition
    - defaults to the current row
3. start column:

- the column number at which the cursor is to first appear
- range is determined by the box definition
- defaults to the current column

4. number of characters:
    - specifies the length of the allowed input
    - actually allows n+1 characters in order to permit alteration of the nth character. The n+1 character is discarded.
    - if the specified number is greater than the space available on the current row, then succeeding rows are used to the limit of that number or the size of the current box
    - defaults to the space available on the current row

5. maximum time:
    - specifies the time allowed before INPUT_KEY will be terminated
    - if expiry occurs then the response made to that time is returned
    - expressed in seconds
    - defaults to system time

6. Input area mark:
    - specifies how the input area is to be marked
    - can be UNDERSCORE, HYPHEN, ASTERISK or blank
    - the underscore, hyphen or asterisk length will be equal to the number of characters parameter
    - defaults to no input area mark displayed

6.3.1 Four Examples Of INPUT_KEY Statements With Associated Displays

Example 1

INPUT_KEY, 1, , , , ,

box number:	1
start row:	defaulted to current row
start column:	defaulted to current column
number of characters:	defaulted to space left on row
maximum time:	defaulted to system time
display area mark:	defaulted to no

Example 2

INPUT_KEY, 2, , , 10, 60, HYPHEN

box number: 2

start row:	defaulted to current row
start column:	defaulted to current column
number of characters:	1 0
maximum time:	60 seconds

display area mark: yes, hyphen

Example 3

INPUT_KEY, 3, , , 1000, , UNDERSCORE

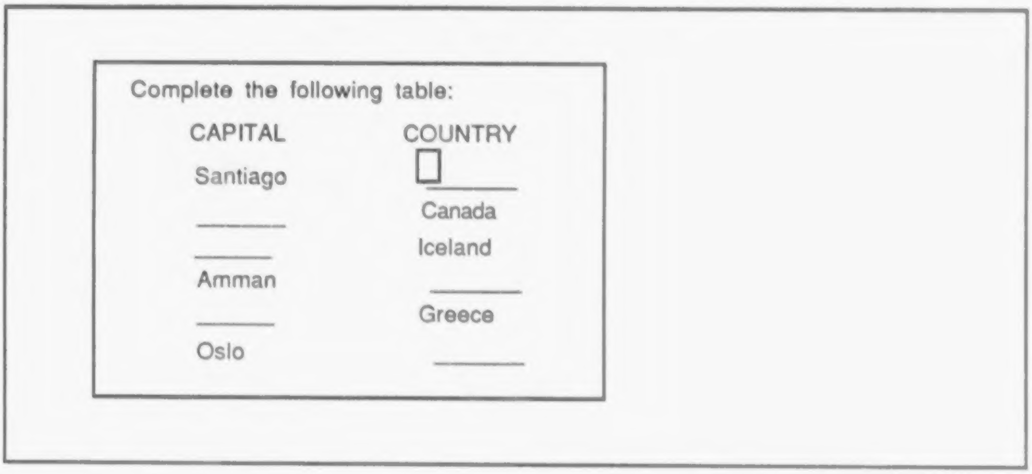
box number: 3
start row: defaulted to current row
start column: defaulted to current column
number of characters: the lesser of 1000 and the space remaining in box
maximum time: defaulted to system time
display area mark: yes, underscore

Example 4

INPUT_KEY, 4, 11, 20, 8, 60,

box number: 4
start row: 11
start column: 20
number of characters: 8
maximum time: 60 seconds
display area mark: defaulted to no

See Figure C6.2 below.



Complete the following table:

CAPITAL	COUNTRY
Santiago	☐___
___	Canada
___	Iceland
Amman	___
___	Greece
Oslo	___

Figure C6.2 Example 4 of INPUT_KEY

6.3.2 Limitations And Shortcomings Of INPUT_PICK

1. INPUT_PICK accommodates a maximum of 100 marked locations but only 10 in each direction - horizontal and vertical. (See arrays called "rows" and "cols".) Since only single character inputs are possible with INPUT_PICK an additional limitation applies if identifiers are used with the choices: a maximum of 9 choices can be accommodated if the identifiers are numeric or 26 choices if they are alphabetic.

2. A maximum of 20 selections can be accommodated on a multiple selection INPUT_PICK. This includes selection made and then subsequently unselected. See the arrays answer row and answer_col.

3. Type ahead is yet to be disabled/enabled correctly;it should probably remain disabled until the call to INPUT_C and disabled again after return.

4. Key-repeat is disabled.

5. The open box is the only choice pointer presently available. It is good for choices displayed with an identifier if the marked location is the same as that of the identifier since the box then nicely surrounds the number or letter. For choices displayed without an identifier a right pointing arrow occupying just one character space would be best.

6. For single entry INPUT_PICKs no marking of a choice is made when it is selected. For multiple selection INPUT_PICKs, as a choice is selected a star is displayed in the marked location to indicate that the selection was made.

7. Maximum time is presently ignored by INPUT_C.INPUT includes the code to accommodate a time out condition but does not return the latency time.

8. Though an options parameter for INPUT_C is defined at <set_input_option> it is not yet implemented.

9. No check is made for the illogical condition of the permitted number of selections exceeding the number of displayed choices.

10. No check is made for proper matching between the choice identifier and the "additional keys available" parameter. Presently an author could specify, for example, numbers for the former and "LETTERS_L_CASE" for the latter.

11. Incorrect identification of choices, such as assigning the same identifier to two choices, will go undetected by INPUT and will not prevent execution.

12. Incorrect specification of marked locations, such as assigning the same matrix position to two choices, will go undetected by DISPLAY but will be caught by INPUT_PICK.

13. Two dimensional multiple choice displays presently require an equal number of marked locations in each row. Completing a display with a marked location but no actual choice will allow execution, though it may be unsatisfactory. Selection of such an invisible choice is not preventable by INPUT_PICK though it could be detected by ANSWER.

14. Upper case alphabetic identifiers sometimes appear better in the display than lower case ones but use of them as additional available keys seems an unnecessary restriction. For example, it seems natural to use the "b" key to select the "B" choice. In such a case it may be useful to allow use of both shifted and nonshifted alphabetic keys for pointer movement. This could be done with the addition of another identifier type, LETTERS_B_CASE.

15. The third parameter of the INPUT_PICK, number of choices, could be defaultable for one dimensional displays.

16. A multiple selection INPUT_PICK becomes complete only after the number of selections made equals the number specified by the author. Early termination is also possible however by typing ! (shifted 1). A more suitable key is needed. Should the possibility of termination should be made optional?

6.4 Limitations And Shortcomings Of INPUT_KEY

1. Type ahead is yet to be disabled/enabled correctly; it should probably remain disabled until the keyboard is activated and disabled again after RETURN is pressed.

2. Maximum time is presently ignored.

3. On multiple row responses the DELETE key erases only to the start of the current row. Though the BACKSPACE key can be used to restart the entire response it would be better on multiple row responses to let the user delete back into the right most column of the previous row.

4. Any character is lost. Whatever is displayed is saved in the response array.

5. Response length is presently limited to 100 characters per row to a maximum of 30 rows. See the arrays row_size and response. The variables row_count and mid_row_length (as determined in <set_row_count>) could be used to set the size of both arrays.

6. Two special characters require special treatment, the back slash and the double quote. When either is included in a response any characters following are not displayed by DISPLAY though they are saved correctly by INPUT_KEY in the response string.

7. If the start location is defaulted and the associated question fills the box, DISPLAY incorrectly sets the next available row to the last text row.

8. The start location can be specified such that it falls within displayed text. Execution can continue but replacement of the displayed text occurs.

9. Presently, it's not possible to force an automatic entry after a single character is typed. If 1 is specified as the maximum number of key strokes, actually 2 must be entered.

6.5 Storage Of Marked Locations For INPUT_PICK

The locations marked in the ware and saved by DISPLAY are retrieved by INPUT and saved in two matrices called rows and cols. The size of these matrices imposes three limitations:

- a maximum of 100 marked locations;
- a maximum of 10 locations displayed either vertically or horizontally
- a maximum of 10 multiple selections.

The optional home location will be saved in 0,0. The following diagram illustrates the storage of the marked locations for use by INPUT_PICK. It shows the rows and columns of the marked locations of the one dimension, 8 choice menu display of the first INPUT_PICK example above.

The two matrices are merged into one diagram shown in Figure C6.3. Within each occupied cell the row number is in the top left while the column is in the bottom right. Thus the first marked location is at row 18 column 6 and the last is row 4, column 6. The rows and columns of a one dimensional menu, whether displayed vertically or horizontally, are saved in this fashion. Blank cells represent a value of zero.

	0	1	2	3	4	5	6	7	8	9	10
0											
1		18 6	16 6	14 6	12 6	10 6	8 6	6 6	4 6		
2											
10											

Figure C6.3 Rows and Columns of Marked Locations

After a choice is selected its display location is saved in position 1,1 of each of the two matrices. Upon completion of INPUT_PICK this information is available to the ANSWER interpreter.

Figure C7.2 illustrates the storage of the marked locations of a two dimensional display. It represents the fourth INPUT_PICK example above in which the menu consists of five rows of four columns each. A home location was also specified and was saved in 0,0. Only the occupied portions of the matrix are shown here.

	0	1	2	3	4	
0	2 55					
1		6 4	6 21	6 40	6 57	
2		8 4	8 19	8 39	8 56	
3		10 4	10 20	10 38	10 57	
4		12 3	12 21	12 39	12 58	
5		14 4	14 21	14 40	14 58	

Figure C6.4 Five Rows and 4 Columns Example

Multiple selection INPUT_PICKs save the first selection in 1,1, the second in 1,2, the third in 1,3, ... 1,10 .Deleted selections are ignored and not returned to the calling grammar

6.6 Glossary

1. area mark - the string of underscores, hyphens or asterisks optionally displayed to indicate the input area
2. choice - a displayed menu option
3. flag - the special character displayed in the marked location of a selected choice to indicate that the selection was recognized. Presently a "*" is used as the character. Flagging is activated on multiple selection INPUT_PICKs only. When a choice is unselected, the flag is replaced with whatever occupied the space previously.
4. home location - the one marked location separate from those of the displayed choices. If it is specified, the pointer appears there on entry into INPUT_PICK. Home location is intended to reduce any biasing toward the topmost displayed choice since that is where the pointer will first appear if a home location is not specified. It is not a selectable choice so RETURN is not active for it.
5. identifier - a number or letter displayed with a choice. If a valid one is typed, the pointer will immediately move to the appropriate choice.
6. key set - the set of keys, in addition to the arrow keys, made available for pointer movement. The set is selected from one of the numbers 1-9, the letters a-z, or the letters A-Z. A set will include only as many keys as there are displayed choices.

7. marked locations - the box locations defined in the ware which are used by INPUT_PICK as pointer positions. Normally there will be one marked location for each choice and it will be alongside that choice.
8. multiple selection - the INPUT_PICK type in which more than one choice must be selected
9. overflow - entry of more characters on a row than the box size allows during execution of a multiple row INPUT_KEY. The one character causing overflow is discarded.
10. pointer - the character displayed in the marked location and which is moved from choice to choice in order to permit a selection.
11. select - the choosing of a displayed choice with the RETURN key

Section C7

Menu Language
Syntax

7.1 Introduction

The menu language is provided for the structuring of menu modules. The menu language facilitates the design of menus and sub-menus to allow students to select choices.They may be choices in a table of contents or a glossary, or multiple-choice type questions in exams, drills or tutorials.

Menu statements may also appear as lines between a MENU line and an END_MENU line in a CONTENT module.

7.2 Syntax Overview

```
<menu_process>
        is      "MENU" <identifier> <eol>
                 [parameter]...
                 [local_constant]...
                 [local_variable]...
                 [declare_statment]
                 [boxstatement]
                 [header_statement]
                 <choices_statement>
                 [choices_statement]...
                 <select_statement>
                 [select_statement]...
                 "END_MENU" <identifier> <eol> ;

<parameter>
        is      <value_parameter>
        or       <reference_parameter> ;

<value_parameter>
        is      "VAL" <identifier> ":"
                 <value_type> <eol> ;

<value_type>
        is      "NUMERIC"
        or      "STRING" "OF" <numeric_expression> ;

<reference_parameter>
        is      "REF" <identifier> ":"
                 <reference_type> <eol> ;

<reference_type>
        is      "NUMERIC"
        or      "STRING"
        or      "LIST"
        or    "RECORD"
        or      <type_identifier> ;

<type_identifier>
        is      <identifier> ;
```

7.3 Initial Syntax Common To Other Interpreters

```
<local_constant>
        is      "CON" <identifier> "="
                 <value> <eol> ;

<value>
         is     /* See Section C2 */ ;

<local_variable>
        is      "VAR" <identifier> ":"
                 <type> <eol> ;

<type>
        is      /* See Section C1 */ ;

<eol>
        is      [comment] *eol ;

<comment>
        is       "///"
                 0 or more of anything ;
```

7.4 Declaration Statement

The declare statement with its associated phrases provides the author with the means of documenting the menu.

```
<declare_statement>
        is     "DECLARE"
                 <eol>
                 [declare_phrases]...   ;

<declare_phrases>
        is     <author_phrase>
        or     <date_phrase>
        or     <comment> <eol> ;

<author_phrase>
        is     "AUTHOR" <author> <eol> ;

<author>
        is      <string_expression>  ;

<date_phrase>
```

```
        is    "DATE" <date> <eol> ;

<date>
        is      <string_expression>  ;
```

7.5 Box Statement

The box statement is used to declare which box of the current screen that the menu will appear in. If a box statement is omitted from the menu, then by default the menu will appear in BOX 0 of the current screen.

```
<box_statement>
        is    "MBOX"
                <box_number>
                <eol> ;

<box_number>
        is      <numeric_expression> ;
```

7.6 Header Statement

The header statement which is optional allows the author to present information or instructions to the menu user. The information or instructions are specified in a manner similar to the TEXT statement in the Display language.

```
<header_statement>
        is    "HEADER"   [format_or_as_is_mode]  <eol>
                [text_line]...   ;

<format_or_as_is_mode>
        is    "FORMAT"
        or       "AS-IS"  /* default */  ;

<text_line>
        is      /* see Section C4 */     ;
```

7.7 Choices Statement

The choices statement provides the author with the means to specify a list or a series of text phrases which will become the available choices for the menu user. More than 1 choices statement can be entered allowing the author to place more than 1 set of choices on the same menu. Which set of choices are active will depend upon the SELECT statement(s).

```
<choices_statement>
        is    "CHOICES"
                <eol>
                button_phrase]
                [size_phrase]
                <list_or_text_phrase>   ;
```

7.7.1 Button Phrase

The button phrase permits the author to prefix each choice with an incrementing index (integer or letter). As well the author can specify the display attributes of the button.

```
<button_phrase>
        is    "BUTTON"
                [display_attribute]
                <initial_button_value>
                <eol> ;
```

The display attribute allows the author to specify how the button should be presented i.e. color, normal or inverted video etc.

```
<display_attribute>
        is      "("
                <display_attribute_string>
                ")"  ;

<display_attribute_string>
        is      <string_expression> ;

<initial_button_value>
        is      <numeric_expression>
        or
                /* if the initial button value is of type string then only the 1st character
                of the string expression will be used. Note also that it must be an upper
                or lower case letter */
                <string_expression>  ;
```

7.7.2 Size Phrase

The Menu interpreter will utilize the space remaining in a display box after the Header statement. However, the author can specify with the AREA phrase how many rows and columns of choices will displayed at a time. As well, should choices occupy more than 1 line of text, the spacing feature is available. Where choices in a list are lengthy, the width feature of COLUMNS allows for truncation to enable more choices to be displayed simultaneously.

```
<size_phrase>
        is    "AREA"
                [row_terms]
                [column_terms]
                <eol> ;

<row_terms>
        is      "ROW"|"S"|
                [no_of_display_rows_expression]
                [display_rows_spacing_expression]
                /* default 1 */  ;
```

The default number of display rows is the number of rows remaining in the current display box after the header information is presented. From this value subtract 2 to allow for a display area at the bottom of a box for the PROMPT, the KEYBOARD response and a possible UNREC message (see SELECT below).

```
<no_of_display_rows_expression>
        is      <numeric_expression>  ;

<display_rows_spacing_expression>
        is      ","
                 <numeric_expression>  ;

<column_terms>
        is      "COL"|"UMNS"|
                 [no_of_display_columns_expression]
                 /* default 1 */
                 [display_column_width_expression]   ;

<no_of_display_columns_expression>
        is      <numeric_expression>  ;
```

If the column display width is defaulted the Menu interpreter will attempt to use as wide a column as possible given the number of display columns and the width of the current box.

```
<display_column_width_expression>
        is      ","
                 <numeric_expression>  ;
```

7.7.3 List Or Text Phrase

Both the LIST phrase and the TEXT phrase allow for the optional specification of display attributes. This allows the author to specify how the choices should be presented i.e. color, normal or inverted video etc., however, position formatting attributes should be avoided. It is possible that the choices will themselves contain display attribute information which will override all other specifications.

```
<list_or_text_phrase>
        is      "LIST"
                 [display_attribute]
                 <qualified_data_element>
                 /* this must point to a store element  of type LIST */
                 <eol>
        or      "TEXT".
                 [display_attribute]   <eol>
                 [text_line]...
                 /* refer to Section C4 */  ;
```

7.8 Select Statement

One or more SELECT statements conclude the body of a menu description. Each SELECT statement requires the menu user to make a choice via the keyboard and/or the pick mechanism. This choice will be returned to the menu caller via a parameter.

```
<select_statement>
        is    "SELECT"
                <return_choice_made_phrase>
                [activate_set_of_choices] /* default 1 */
                <eol>
                [select_options]...  ;

<return_choice_made_phrase>
        is       [return_type_description]
                <choice_made_parameter>  ;
```

7.8.1 Return Choice Made Parameter

In the default situation, given that a Button has been specified in the Choices statement, then the value of the button will be returned to the menu caller via the return parameter. If a button has not been specified then the complete text of the choice will be returned.

```
<return_type_description>
        is       "("
                [return_type_description_value] ")"    ;

<return_type_description_value>
        is     "C"|"HARACTERS"|
                [return_type_description_quantity]
        or      "W"|"ORDS"|
                [return_type_description_quantity]
        or      "B"|"UTTON"|   ;

<return_type_description_quantity>
        is      <numeric_expression>  ;
                <choice_made_parameter>is
                <qualified_data_element>  ;
```

7.8.2 Activate Selects

If the menu author specifies more than one CHOICES statement then the author must specify which CHOICES statements are to be active with the current SELECT statement and in what combination. The interpreter numerates the CHOICES statements sequentially from 1. The author can indicate that only one of the CHOICES list is to be active by specifying the number for that CHOICE.

Alternatively the author could combine lists by specifying "1 AND 2" etc. which means that all choices in list 1 and list 2 are allowable. The last option for the author is to specify that multiple lists are allowable but that the user must utilize KEYPAD keys to move from list to list. An example of this syntax would be "1 OR 2 OR 4". Note that the author in specifying "AND" or "OR" is constrained to only one of the two keywords within a list activation

```
phrase.<activate_set_of_choices>
        is     <choices_selection>
                 [more_choices_selection]  ;

<more_choices_selection>
        is     <and_choices_selection>
        or      <or_choices_selection> ;

<choices_selection>
        is      <numeric_expression> ;

<and_choices_selection>
        is     "AND"
                 <choices_selection>
                 [and_choices_selection]  ;

<or_choices_selection>
        is     "OR"
                 <choices_selection>
                 [or_choices_selection]  ;
```

7.8.3 Select Options

```
<select_options>
        is     "TIME"        <select_time_option> <eol>
        or      "PROMPT"       <select_prompt_option>
                 <eol>
        or     "KEYBOARD"    <eol>
        or      "PICK"       <eol>
        or      "UNREC"       <select_unrec_option>
                 <eol> ;
```

The lack of a TIME phrase in the SELECT statement indicates the user has an unrestricted amount of time to respond. If a TIME phrase is present and the the user exceeds the allotted time, then the UNREC message will be presented. If no UNREC phrase is contained in the SELECT statement, then a MENU system message will be presented to the user on the bottom line of the screen.

```
<select_time_option>
        is      <numeric_expression> ;

<select_prompt_option>
        is      <string_expression> ;

<select_unrec_option>
        is      <string_expression> ;
```

7.9 Examples

7.9.1 Example 1

```
HEADER
At this point the menu creator can insert any information he/she deems required. Note
however, that it must end ala a TEXT statement in DISPLAY subsystem.\e
CHOICES
   BUTTON ("\cy") 1
        TEXT ("\cg")
Continue with my course
Review examination marks
End this session\e
SELECT return_buf
        PROMPT "Enter your choice!"
```

7.9.2 Example 2

```
HEADER
At this point the menu creator can insert any information deemed required. Note
however, that it must end ala TEXT statement in DISPLAY subsystem.\e
CHOICES
        TEXT
Continue with my course
\C  this indicates a continuation of choice text
Review examination marks
End this session\e
SELECT (3 CHARACTERS) return_buffer
        PROMPT "Key in first 3 letters of your choice!"
```

7.9.3 Example 3

```
HEADER AS-IS
                                CAI USER           13-AUG-1985
Name:\v(last_name), \v(first_name)      Type:\v(type)     User
Id:\v(user_id)
Cai Course:\v(course) Class name:\v(class_name)\e
CHOICES
        BUTTON("\cy\i")  1
        TEXT("\cg")
Continue with my course
Review my examination marks
End this session\e
SELECT  response_buffer
        PROMPT  "Indicate your choice"
```

7.9.4 Example 4

```
HEADER
Experimental ERAS System. ARD
Select one of the following options\e
CHOICES
        BUTTON("\cc")  1
        TEXT("\cg")
Edit a database
Initialize a database
Create a database
Exit from this process\e
SELECT( 1 WORD ) command_buffer
```

```
          PICK
          KEYBOARD
          TIME=60
          UNREC "\crDid not recognize selection!"
```

7.9.5 Example 5

```
HEADER AS-IS
          USER SURNAMES               OPTIONS\e
CHOICES
          AREA ROW 6 COL 1,25
          LIST ("\cc") user_surnames
CHOICES
          BUTTON ("\cy") 1
          TEXT ("\cy" )
change exam mark
read mail from user
send mail to user
review position in course
halt user
SELECT (1 WORD ) user_name_buffer 1
          PROMPT "Which user?"
          UNREC "User name not in current list "
SELECT ( BUTTON ) command_buffer 2
          PROMPT "Please indicate an option"
```

Section C8

Expressions Syntax

8.1 Introduction

This chapter briefly outlines the facilities currently available in expressions. In the following discussion, simplified syntactic specifications will be used. Items in lower case are syntactic element descriptors. Items in upper case are actual symbols to be used. Syntactic elements may be grouped by enclosing them in braces, {... }, indicating that they are required; or in brackets, [...], indicating that they are optionai. If the right brace is followed by an ellipsis, }..., then the enclosed syntactic elements may be repeated but must appear at least once. If the right bracket is followed by an ellipsis,]..., then the enclosed syntactic elements may be repeated zero or more times. If a brace or a bracket is a syntactic element it will be enclosed in quotation marks, e.i. "{","}", "[", "]". The vertical bar, |, separates groups of alternate syntactic elements. All other symbols should be considered syntactic elements.

8.2 The Assignment Statement

The basic form is:

qualified_data_element assignment_operator expression

Numeric assignment operators are:

:=	assign numeric value
:+	increment numeric value
:-	decrement numeric value
:C=	convert numeric in string to numeric value

String assignment operators are:

:=	assign string value
:+	append with no intervening space
:&	append with one intervening space
:C=	convert numeric value to its string representation

The assignment operators were chosen because they are very common in structured languages, e.i., ALGOL, COMAL, Modula2, and Pascal.

8.3 String Expressions

The form of a string expression is:

string_term [string_operator string_term]...

The form of a string term is:

string_factor [string_multiplier]

The form of a string multiplier is:

* numeric_expression

The effect of the optional string multiplier is to concatenate a number of copies of the string factor, e.g.,

"fred" * 3 is "fredfredfred"

The string operators are:

+	concatenate with no intervening space
&	concatenate with one intervening space

The form of a string factor is:

```
string_value
qualified_data_element
!user_string_function
&eras_string_function
( string_expression )
```

A string value is 0 or more printable characters enclosed in double quotation marks. The quotation mark is itself represented by two successive quotation marks. For example,the string assignment:

```
str := "John called out, ""Hi Mom!"""
```

would assign the following string literal to variable 'str':

```
John called out, "Hi Mom!"
```

The string qualified data element and the string function call may be followed by an optional subpart specification. The form of the subpart specification is:

```
"{" [ left_offset ] [ ^ right_offset | ~ length ] "}"
```

Left offset, right offset, and length are numeric expressions. A special system variable, $, representing the current length of the string being subparted, may participate in these numeric expressions. The left offset defaults to 0 and the right part defaults to ^ $-1. That is, they follow the same rules as Elf subparts but the numbers may be numeric expressions.

8.4 String Relations

In the definition of a numeric expression that follows, at the relational operator precedence level a string relation may be substituted for a series of numeric relations. That is, the following two statements are equivalent as far as their precedence level participation in a numeric expression is concerned:

```
arithmetic_exp { relational_operator arithmetic_exp }...
string_exp string_relational_operator string_exp
```

There is an additional string relational operator besides the normal relational operators. The form is as follows:

string_exp_1 IN string_exp_2

This relation returns the position (plus 1) of string_exp_1in string_exp_2. If string_exp_1 is not in string_exp_2, it returns 0. (Remember in ERAS that 0 is FALSE and anything other than 0 is TRUE.)

8.5 Numeric Expressions

The general form of a numeric expression is as follows:

operand [operator operand]...

The special exception is the participation of a string relation at the relational operator precedence level. The order of precedence of numeric operators is as follows:

1. * / DIV MOD
2. + -
3. unary + or -
4. = <> < > <= >=
5. unary NOT
6. AND
7. OR XOR

DIV is integer division and MOD is remainder after integer division.

The form of a numeric operand is as follows:

numeric_value
boolean_value
qualified_data_element
!user_numeric_function
&eras_numeric_function
(numeric_expression)

The following are equivalent boolean values that equal 1.000:

TRUE YES ON 1

The following are equivalent boolean values that equal 0:

FALSE NO OFF 0

The form for a numeric value is:

[sign] whole_number_part [. fractional_part](NOTE: no spaces allowed within number)
sign is + or -
whole_number_part between 0 and 2147482, inc.
fractional_part between 0 and 999, inc.

All numerics are stored as integers equal to the numeric times 1000.

8.6 Qualified Data Element

The form of a qualified data element is as follows:
variable_identifier [list_index]... [field_reference]...

The form of a field reference is:

.field_identifier [list_index]...

(NOTE: the dot must exactly precede the field identifier)

The form of a list index is:

"[" location [, location]... "]"

A location may be a numeric expression or one of the following key words:
FIRST LAST NEXT PREV EOL BOL EMPTY

The first four are used to obtain the referenced element in the list. NEXT and PREV have no meaning until the list has been entered by FIRST or LAST. If no such element exists, an error is generated. The last three return a boolean result for the referenced list, i.e.

EOL - is the list pointer at the End Of the List
BOL - is the list pointer at the Beginning Of the List
EMPTY - is the list empty